The Ames Stereo Pipeline:

NASA's Open Source Automated Stereogrammetry Software

A part of the NASA NeoGeography Toolkit

Version 2.4.2

Intelligent Robotics Group
NASA Ames Research Center
stereo-pipeline-owner@lists.nasa.gov

October 5, 2014

Credits

This open source version of the Ames Stereo Pipeline (ASP) was developed by the Intelligent Robotics Group (IRG), in the Intelligent Systems Division at the National Aeronautics and Space Administration (NASA) Ames Research Center in Moffett Field, CA. It builds on over ten years of IRG experience developing surface reconstruction tools for terrestrial robotic field tests and planetary exploration.

Project Lead

- Zachary Moratto (NASA/Stinger-Ghaffarian Technologies)
  `z.m.moratto@nasa.gov`

Development Team

- Oleg Alexandrov (NASA/Stinger-Ghaffarian Technologies)
- Scott McMichael (NASA/Stinger-Ghaffarian Technologies)
- Dr. Ross Beyer (NASA/SETI Institute)

Former Developers

- Dr. Ara Nefian (NASA/Carnegie Mellon University)
- Matthew Hancher (NASA)
- Mike Lundy (NASA/Stinger-Ghaffarian Technologies)
- Vinh To (NASA/Stinger-Ghaffarian Technologies)

Contributing Developer & Former Principal Investigator (LMMP)

- Michael J. Broxton (NASA/Carnegie Mellon University)

Contributing Developer & Former IRG Terrain Reconstruction Lead

- Dr. Laurence Edwards (NASA)

A number of student interns have made significant contributions to this project over the years: Kyle Husmann (California Polytechnic State University), Sasha Aravkin (Washington State University), Aleksandr Segal (Stanford), Patrick Mihelich (Stanford University), Melissa Bunte (Arizona State University), Matthew Faulkner (Massachusetts Institute of Technology), Todd Templeton (UC Berkeley), Morgon Kanter (Bard College), Kerri Cahoy (Stanford University), and Ian Saxton (UC San Diego).

The open source Stereo Pipeline leverages stereo image processing work, past and present, led by Michael J. Broxton (NASA/CMU), Dr. Laurence Edwards (NASA), Eric Zbinden (formerly NASA/QSS Inc.), Dr. Michael Sims (NASA), and others in the Intelligent Systems Division at NASA Ames Research Center. It has benefited substantially from the contributions of Dr. Keith Nishihara (formerly NASA/Stanford), Randy Sargent (NASA/Carnegie Mellon University), Dr. Judd Bowman (formerly NASA/QSS Inc.), Clay Kunz (formerly NASA/QSS Inc.), and Dr. Matthew Deans (NASA).

Acknowledgments

The initial adaptation of Ames' stereo surface reconstruction tools to orbital imagers was a result of a NASA funded, industry led project to develop automated digital elevation model (DEM) generation techniques for the Mars Global Surveyor (MGS) mission. Our work with that project's Principal Investigator, Dr. Michael Malin of Malin Space Science Systems (MSSS), and Co-Investigator, Dr. Laurence Edwards of NASA Ames, inspired the idea of making stereo surface reconstruction technology available and accessible to a broader community. We thank Dr. Malin and Dr. Edwards for providing the initial impetus that in no small way made this open source stereo pipeline possible, and we thank Dr. Michael Caplinger, Joe Fahle and others at MSSS for their help and technical assistance.

We'd also like to thank our friends and collaborators Dr. Randolph Kirk, Dr. Brent Archinal, Trent Hare, and Mark Rosiek of the United States Geological Survey's (USGS's) Astrogeology Science Center in Flagstaff, AZ, for their encouragement and willingness to share their experience and expertise by answering many of our technical questions. We also thank them for their ongoing support and efforts to help us evaluate our work. Thanks also to the USGS Integrated Software for Imagers and Spectrometers (ISIS) team, especially Jeff Anderson and Kris Becker, for their help in integrating stereo pipeline with the USGS ISIS software package.

Thanks go also to Dr. Mark Robinson, Jacob Danton, Ernest Bowman-Cisneros, Dr. Sam Laurence, and Melissa Bunte at Arizona State University for their help in adapting the Ames Stereo Pipeline to lunar data sets including the Apollo Metric Camera.

We'd also like to thank David Shean, Dr. Ben Smith, and Dr. Ian Joughin of the Applied Physics Laboratory at the University of Washington for providing design direction for adapting Ames Stereo Pipeline to Earth sciences and in particular the Digital Globe mode.

Finally, we thank Dr. Ara Nefian, and Dr. Laurence Edwards for their contributions to this documentation, and Dr. Terry Fong (IRG Group Lead) for his management and support of the open source and public software release process.

Portions of this software were developed with support from the following NASA Science Mission Directorate (SMD) and Exploration Systems Mission Directorate (ESMD) funding sources: the Mars Technology Program, the Mars Critical Data Products Initiative, the Mars Reconnaissance Orbiter mission, the Applied Information Systems Research program grant #06-AISRP06-0142, the Lunar Advanced Science and Exploration Research (LASER) program grants #07-LASER07-0148 and #11-LASER11-0112, the ESMD Lunar Mapping and Modeling Program (LMMP), and the SMD Cryosphere Program.

Contents

1 Introduction **1**

1.1 Background 1

1.2 Human vs. Computer: When to Choose Automation? 2

1.3 Software Foundations 3

1.3.1 NASA Vision Workbench 3

1.3.2 The USGS Integrated Software for Imagers and Spectrometers 3

1.4 Getting Help 4

1.5 How to File Bug Reports 4

1.6 Typographical Conventions 4

1.7 Referencing the Ames Stereo Pipeline in Your Work 5

1.8 Warnings to Users of the Ames Stereo Pipeline 5

I Getting Started **7**

2 Installation **9**

2.1 Binary Installation 9

2.1.1 Quick Start for ISIS Users 9

2.1.2 Quick Start for Digital Globe Users 10

2.1.3 Common Errors 10

2.2 Installation from Source 11

2.3 Settings Optimization 11

2.3.1 Performance Settings 13

2.3.2 Logging Settings 13

3 Tutorial: Processing Mars Orbiter Camera Imagery **15**

3.1 Quick Start 15

3.2 Preparing the Data 15

3.2.1 Loading and Calibrating Images using ISIS 16

3.2.2 Aligning Images 16

4 Tutorial: Processing Earth Digital Globe Imagery **19**

4.1 Processing Raw 20

4.2 Processing Map-Projected Imagery 21

4.3 Handling CCD Boundary Artifacts 22

4.4 Dealing with Terrain Lacking Large Scale Features 23

4.5 Processing Multi-Spectral Images 24

5 The Next Steps **25**

5.1 Stereo Pipeline in More Detail 25

5.1.1 Setting Options in the `stereo.default` File 25

5.1.2 Performing Stereo Correlation 27

5.1.3 Specifying Settings on the Command Line 27

5.1.4 Stereo on Multiple Machines 28

5.1.5 Multi-View Stereo 28

5.1.6 Diagnosing Problems 28

5.2 Visualizing and Manipulating the Results 30

5.2.1 Building a 3D Model 30

5.2.2 Building a Digital Elevation Model 30

5.2.3 Fine-Tuning the Results 31

5.2.4 Alignment to Point Clouds From a Different Source 33

5.2.5 Creating DEMs Relative to the Geoid/Areoid 34

5.2.6 Converting to the LAS Format 34

5.2.7 Generating Color Hillshade Maps 34

5.2.8 Building Overlays for Moon and Mars Mode in Google Earth 34

II The Stereo Pipeline in Depth **37**

6 Stereo Correlation **39**

6.1 Pre-Processing 39

6.2 Disparity Map Initialization 41

6.2.1 Debugging Disparity Map Initialization 42

6.2.2 Local Homography 43

6.3 Sub-pixel Refinement 43

6.4 Triangulation 45

7 **Bundle Adjustment** **47**

7.1 Overview . . . 47

7.2 Bundle adjustment using ASP . . . 48

7.3 Bundle adjustment using ISIS . . . 48

7.3.1 Tutorial: Processing Mars Orbital Camera Imagery . . . 50

8 **Data Processing Examples** **55**

8.1 Guidelines for Selecting Stereo Pairs . . . 55

8.1.1 Combating Long Run Times . . . 55

8.2 Mars Reconnaissance Orbiter HiRISE . . . 56

8.2.1 Columbia Hills . . . 57

8.3 Mars Reconnaissance Orbiter CTX . . . 58

8.3.1 North Terra Meridiani . . . 59

8.4 Mars Global Surveyor MOC-NA . . . 60

8.4.1 Ceraunius Tholus . . . 60

8.5 Mars Exploration Rovers MER . . . 61

8.5.1 PANCAM, NAVCAM, HAZCAM . . . 61

8.6 Lunar Reconnaissance Orbiter LROC NAC . . . 63

8.6.1 Lee-Lincoln Scarp . . . 63

8.7 Apollo 15 Metric Camera Images . . . 64

8.7.1 Ansgarius C . . . 64

8.8 Cassini ISS NAC . . . 66

8.8.1 Rhea . . . 66

8.9 Digital Globe Imagery . . . 68

8.10 GeoEye and Astrium Imagery / RPC Imagery . . . 68

8.11 Dawn (FC) Framing Camera . . . 69

III Appendices **71**

A Tools **73**

A.1 stereo . . . 73

A.1.1 Entry Points . . . 74

A.1.2 Decomposition of Stereo . . . 74

A.2 parallel_stereo . . . 75

A.2.1 Advanced usage . . . 76

A.3 bundle_adjust . . . 77

A.4 point2dem . . . 79
A.4.1 Comparing with MOLA Data . . . 79
A.4.2 Post Spacing . . . 80
A.4.3 Using with LAS or CSV Clouds . . . 80
A.5 point2mesh . . . 83
A.6 dem_mosaic . . . 85
A.7 dem_geoid . . . 87
A.8 dg_mosaic . . . 87
A.9 mapproject . . . 89
A.10 disparitydebug . . . 90
A.11 orbitviz . . . 90
A.12 cam2map4stereo.py . . . 92
A.13 point2las . . . 93
A.14 pc_align . . . 93
A.15 wv_correct . . . 96
A.16 lronac2mosaic.py . . . 97

B The `stereo.default` File 99
B.1 Preprocessing . . . 99
B.2 Correlation . . . 100
B.3 Subpixel Refinement . . . 102
B.4 Filtering . . . 103
B.5 Post-Processing (Triangulation) . . . 104

C Guide to Output Files 105

Bibliography 107

Chapter 1

Introduction

The NASA Ames Stereo Pipeline (ASP) is a suite of automated geodesy and stereogrammetry tools designed for processing planetary imagery captured from orbiting and landed robotic explorers on other planets or here on Earth. It was designed to process stereo imagery captured by NASA and commercial spacecraft and produce cartographic products including digital elevation models (DEMs), ortho-projected imagery, and 3D models. These data products are suitable for science analysis, mission planning, and public outreach.

1.1 Background

The Intelligent Robotics Group (IRG) at the NASA Ames Research Center has been developing 3D surface reconstruction and visualization capabilities for planetary exploration for more than a decade. First demonstrated during the Mars Pathfinder Mission, the IRG has delivered tools providing these capabilities to the science operations teams of the Mars Polar Lander (MPL) mission, the Mars Exploration Rover (MER) mission, the Mars Reconnaissance Orbiter (MRO) mission, and most recently the Lunar Reconnaissance Orbiter (LRO) mission. A critical component technology enabling this work is the Ames Stereo Pipeline (ASP). The Stereo Pipeline generates high quality, dense, texture-mapped 3D surface models from stereo image pairs.

Although initially developed for ground control and scientific visualization applications, the Stereo Pipeline has evolved in recent years to address orbital stereogrammetry and cartographic applications. In particular, long-range mission planning requires detailed knowledge of planetary topography, and high resolution topography is often derived from stereo pairs captured from orbit. Orbital mapping satellites are sent as precursors to planetary bodies in advance of landers and rovers. They return a wealth of imagery and other data that helps mission planners and scientists identify areas worthy of more detailed study. Topographic information often plays a central role in this planning and analysis process.

Our recent development of the Stereo Pipeline coincides with a period of time when NASA orbital mapping missions are returning orders of magnitude more data than ever before. Data volumes from the Mars and Lunar Reconnaissance Orbiter missions now measure in the tens of terabytes. There is growing consensus that existing processing techniques, which are still extremely human intensive and expensive, are no longer adequate to address the data processing needs of NASA and the Planetary Science community. To pick an example of particular relevance, the High Resolution Imaging Science Experiment (HiRISE) instrument has captured a few thousand stereo pairs. Of these, only about two hundred stereo pairs have been processed to date; mostly on human-operated, high-end photogrammetric workstations. It is clear that much more value could be extracted from this valuable raw data if a more streamlined, efficient process could be developed.

The Stereo Pipeline was designed to address this very need. By applying recent advances in robotics and computer vision, we have created an *automated* process that is capable of generating high quality DEMs

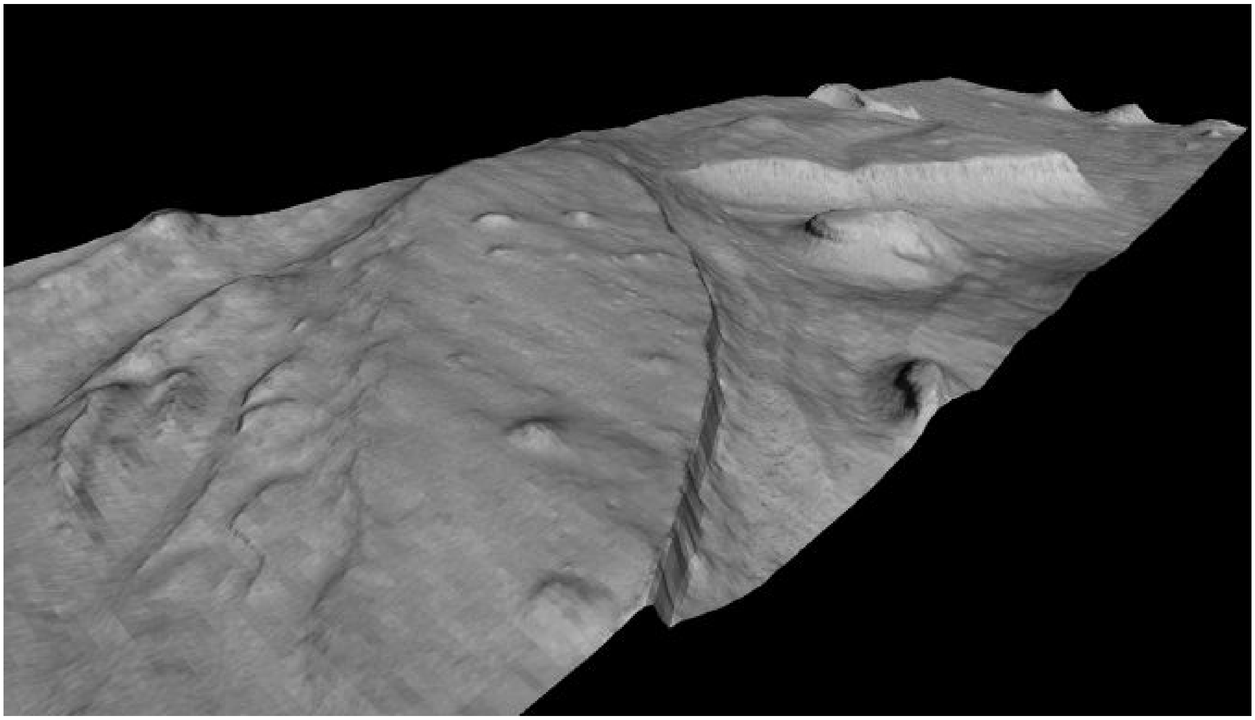


Figure 1.1: This 3D model was generated from a Mars Orbiter Camera (MOC) image pair M01/00115 and E02/01461 (34.66N, 141.29E). The complete stereo reconstruction process takes approximately thirty minutes on a 3.0 GHz workstation for input images of this size (1024×8064 pixels). This model, shown here without vertical exaggeration, is roughly 2 km wide in the cross-track dimension.

with minimal human intervention. Users of the Stereo Pipeline can expect to spend some time picking a handful of settings when they first start processing a new type of imagery, but once this is done, the Stereo Pipeline can be used to process tens, hundreds, or even thousands of stereo pairs without further adjustment. With the release of this software, we hope to encourage the adoption of this tool chain at institutions that run and support these remote sensing missions. Over time, we hope to see this tool incorporated into ground data processing systems alongside other automated image processing pipelines. As this tool continues to mature, we believe that it will be capable of producing digital elevation models of exceptional quality without any human intervention.

1.2 Human vs. Computer: When to Choose Automation?

When is it appropriate to choose automated stereo mapping over the use of a conventional, human-operated photogrammetric workstation? This is a philosophical question with an answer that is likely to evolve over the coming years as automated data processing technologies become more robust and widely adopted. For now, our opinion is that you should *always* rely on human-guided, manual data processing techniques for producing mission critical data products for missions where human lives or considerable capital resources are at risk. In particular, maps for landing site analysis and precision landing absolutely require the benefit of an expert human operator to eliminate obvious errors in the DEMs, and also to guarantee that the proper procedures have been followed to correct satellite telemetry errors so that the data have the best possible geodetic control.

When it comes to using DEMs for scientific analysis, both techniques have their merits. Human-guided stereo reconstruction produces DEMs of unparalleled quality that benefit from the intuition and experience of an expert. The process of building and validating these DEMs is well-established and accepted in the scientific community.

However, only a limited number of DEMs can be processed to this level of quality. For the rest, automated stereo processing can be used to produce DEMs at a fraction of the cost. The results are not necessarily less accurate than those produced by the human operator, but they will not benefit from the same level of scrutiny and quality control. As such, users of these DEMs must be able to identify potential issues, and be on the lookout for errors that may result from the improper use of these tools.

We recommend that all users of the Stereo Pipeline take the time to thoroughly read this documentation and build an understanding of how stereo reconstruction and bundle adjustment can be best used together to produce high quality results. You are welcome to contact us if you have any questions (section 1.4).

1.3 Software Foundations

1.3.1 NASA Vision Workbench

The Stereo Pipeline is built upon the Vision Workbench software which is a general purpose image processing and computer vision library also developed by the IRG. Some of the tools discussed in this document are actually Vision Workbench programs, and any distribution of the Stereo Pipeline requires the Vision Workbench. Unless you're compiling the Vision Workbench and Stereo Pipeline from source, the distinctions probably won't matter to you.

1.3.2 The USGS Integrated Software for Imagers and Spectrometers

For processing non-terrestrial NASA satellite imagery, Stereo Pipeline must be installed alongside a copy of United States Geological Survey (USGS) Integrated Software for Imagers and Spectrometers (ISIS). ISIS is however not required for processing Digital Globe images of Earth, as described in section 2.1.2.

ISIS is widely used in the planetary science community for processing raw spacecraft imagery into high level data products of scientific interest such as map-projected and mosaicked imagery [1, 10, 31]. We chose ISIS because (1) it is widely adopted by the planetary science community, (2) it contains the authoritative collection of geometric camera models for planetary remote sensing instruments, and (3) it is open source software that is easy to leverage.

By installing the Stereo Pipeline, you will be adding an advanced stereo image processing capability that can be used in your existing ISIS workflow. The Stereo Pipeline supports the ISIS "cube" (`.cub`) file format, and can make use of the ISIS camera models and ancillary information (i.e. SPICE kernels) for imagers on many NASA spacecraft. The use of this single standardized set of camera models ensures consistency between products generated in the Stereo Pipeline and those generated by ISIS. Also by leveraging ISIS camera models, the Stereo Pipeline can process stereo pairs captured by just about any NASA mission.

1.4 Getting Help

All bugs, feature requests, and general discussion should be sent to the Ames Stereo Pipeline user mailing list:

stereo-pipeline@lists.nasa.gov

To subscribe to this list, send an empty email message with the subject 'subscribe' (without the quotes) to:

stereo-pipeline-request@lists.nasa.gov

To contact the lead developers and project manager directly, send mail to:

stereo-pipeline-owner@lists.nasa.gov

1.5 How to File Bug Reports

If Stereo Pipeline crashes or produces incorrect results, we would very much like to hear from you. You can send an email to `stereo-pipeline-owner@lists.nasa.gov` describing the problem. It will be helpful to attach the logs output by `stereo` and other tools (section 2.3.2). In some cases we may request your input data as well.

1.6 Typographical Conventions

Names of programs that are meant to be run on the command line are written in a constant-width font, like the `stereo` program, as are options to those programs.

An indented line of constant-width text can be typed into your terminal, these lines will either begin with a '`>`' to denote a regular shell, or with '`ISIS`' which denotes an ISIS-enabled shell (which means you have to set the `ISISROOT` environment variable and sourced the appropriate ISIS 3 Startup script, as detailed in the ISIS 3 instructions).

```
> ls
```

```
ISIS 3> pds2isis
```

Italicized constant-width text denotes an option or argument that a user will need to supply. For example, '`stereo E0201461.map.cub M0100115.map.cub out`' is specific, but '`stereo` *`left-image right-image`* `out`' indicates that *`left-image`* and *`right-image`* are not the names of specific files, but dummy parameters which need to be replaced with actual file names.

Square brackets denote optional options or values to a command, and items separated by a vertical bar are either aliases for each other, or different, specific options. Default arguments are prefixed by an equals sign within parentheses, and line continuation with a backslash:

```
point2dem [--help|-h] [-r moon|mars] [-s float(=0)] \
          [-o output-filename] pointcloud-PC.tif
```

The above indicates a run of the `point2dem` program. The only argument that it requires is a point cloud file, which is produced by the `stereo` program and ends in `-PC.tif`, although its prefix could be anything (hence the italics for that part). Everything else is in square brackets indicating that they are optional.

Both `--help` and `-h` are really the same thing (both will get you help). Similarly, the argument to the `-r` option must be either `moon` or `mars`. The `-s` option takes a floating point value as its argument, and has a default value of zero. The `-o` option takes a filename that will be used as the output DEM.

Although there are two lines of constant-width text, the backslash at the end of the first line indicates that the command continues on the second line. You can either type everything into one long line on your own terminal, or use the backslash character (or appropriate line continuation character) and a return to continue typing on a second line in your terminal.

1.7 Referencing the Ames Stereo Pipeline in Your Work

Although no peer-reviewed paper or report yet exists which details the Ames Stereo Pipeline (see the warning below about this being **research** software), if you do use this software in your work, we'd appreciate it if you referenced one or more of these abstracts:

Moratto, Z. M., M. J. Broxton, R. A. Beyer, M. Lundy, and K. Husmann. 2010. Ames Stereo Pipeline, NASA's Open Source Automated Stereogrammetry Software. *Lunar and Planetary Science Conference* **41**, abstract #2364. [ADS Abstract].

Broxton, M. J. and L. J. Edwards. 2008. The Ames Stereo Pipeline: Automated 3D Surface Reconstruction from Orbital Imagery. *Lunar and Planetary Science Conference* **39**, abstract #2419. [ADS Abstract].

1.8 Warnings to Users of the Ames Stereo Pipeline

Ames Stereo Pipeline is a **research** product. There may be bugs or incomplete features. We reserve the ability to change the API and command line options of the tools we provide. Although we hope you will find this release helpful, you may use it at your own risk. Please check each release's **NEWS** file to see a summary of our recent changes.

While we are confident that the algorithms used by this software are robust, they have not been systematically tested or rigorously compared to other methods in the peer-reviewed literature. We *strongly recommend* that you consult us first before publishing any results based on the cartographic products produced by this software.

Part I

Getting Started

Chapter 2

Installation

2.1 Binary Installation

This is the recommended method. Only the Stereo Pipeline binaries are required. ISIS is required only for users who wish to process NASA non-terrestrial imagery. A full ISIS installation is not required for operation of Stereo Pipeline programs (only the ISIS data directory is needed), but is required for certain preprocessing steps before Stereo Pipeline programs are run for planetary data. If you only want to process terrestrial Digital Globe imagery, skip to section 2.1.2.

Stereo Pipeline Tarball
The main Stereo Pipeline page is `http://irg.arc.nasa.gov/ngt/stereo`. Download the option that matches the platform you wish to use. The recommended, but optional, ISIS version is listed next to the name.

USGS ISIS
If you are working with non-terrestrial imagery, you will need to install ISIS so that you can perform preprocessing such as radiometric calibration and ephemeris attachment. The ISIS installation guide is at `http://isis.astrogeology.usgs.gov/documents/InstallGuide`. You must use their binaries as-is; if you need to recompile, you can follow the *Source Installation* guide for the Stereo Pipeline in Section 2.2. Note also that the USGS provides only the current version of ISIS and the previous version (denoted with a '`_OLD`' suffix) via their `rsync` service. If the current version is newer than the version of ISIS that the Stereo Pipeline is compiled against, be assured that we're working on rolling out a new version. However, since Stereo Pipeline has its own self-contained version of ISIS's libraries built internally, you should be able to use a newer version of ISIS with the now dated version of ASP. This is assuming no major changes have taken place in the data formats or camera models by USGS. At the very least, you should be able to rsync the previous version of ISIS if a break is found. To do so, view the listing of modules that is provided via the '`rsync isisdist.astrogeology.usgs.gov::`' command. You should see several modules listed with the '`_OLD`' suffix. Select the one that is appropriate for your system, and `rsync` according to the instructions.

In closing, running the Stereo Pipeline executables only requires that you have downloaded the ISIS secondary data and have appropriately set the `ISIS3DATA` environment variable. This is normally performed for the user by ISIS startup script, `$ISISROOT/scripts/isis3Startup.sh`.

2.1.1 Quick Start for ISIS Users

Fetch Stereo Pipeline
Download the Stereo Pipeline from `http://irg.arc.nasa.gov/ngt/stereo`.

Fetch ISIS Binaries

As detailed at http://isis.astrogeology.usgs.gov/documents/InstallGuide.

Fetch ISIS Data

As detailed at http://isis.astrogeology.usgs.gov/documents/InstallGuide.

Untar Stereo Pipeline

`tar xzvf StereoPipeline-`*`VERSION-ARCH-OS`*`.tar.gz`

Add Stereo Pipeline to Path (optional)

bash: `export PATH="`*`/path/to/StereoPipeline`*`/bin:${PATH}"`
csh: `setenv PATH "`*`/path/to/StereoPipeline`*`/bin:${PATH}"`

Set Up ISIS

bash:

```
export ISISROOT=/path/to/isisroot
source $ISISROOT/scripts/isis3Startup.sh
```

csh:

```
setenv ISISROOT /path/to/isisroot
source $ISISROOT/scripts/isis3Startup.csh
```

Try It Out

See the next chapter (Chapter 3) for an example.

2.1.2 Quick Start for Digital Globe Users

Fetch Stereo Pipeline

Download the Stereo Pipeline from http://irg.arc.nasa.gov/ngt/stereo.

Untar Stereo Pipeline

`tar xvfz StereoPipeline-`*`VERSION-ARCH-OS`*`.tar.gz`

Try It Out

Processing Earth imagery is described in the data processing tutorial in chapter 4.

2.1.3 Common Errors

Here are some errors you might see, and what it could mean. Treat these as templates for problems. In practice, the error messages might be slightly different.

```
**I/O ERROR** Unable to open [$ISIS3DATA/Some/Path/Here].
Stereo step 0: Preprocessing failed
```

You need to set up your ISIS environment or manually set the correct location for `ISIS3DATA`.

```
point2mesh stereo-output-PC.tif stereo-output-L.tif
[...]
99%  Vertices:   [************************************************************] Complete!
       > size: 82212 vertices
Drawing Triangle Strips
Attaching Texture Data
zsh: bus error  point2mesh stereo-output-PC.tif stereo-output-L.tif
```

The source of this problem is an old version of OpenSceneGraph in your library path. Check your `LD_LIBRARY_PATH` (for Linux), `DYLD_LIBRARY_PATH` (for OSX), or your `DYLD_FALLBACK_LIBRARY_PATH` (for OSX) to see if you have an old version listed, and remove it from the path if that is the case. It is not necessary to remove the old versions from your computer, you just need to remove the reference to them from your library path.

```
bash: stereo: command not found
```

You need to add the `bin` directory of your deployed Stereo Pipeline installation to the environmental variable `PATH`.

2.2 Installation from Source

This method is for advanced users. You will need to fetch the Stereo Pipeline source code from GitHub at https://github.com/NeoGeographyToolkit/StereoPipeline and then follow the instructions specified in `INSTALLGUIDE`.

2.3 Settings Optimization

Finally, the last thing to be done for Stereo Pipeline is to setup up Vision Workbench's render and logging settings. This step is optional, but for best performance some thought should be applied here.

Vision Workbench is a multithreaded image processing library used by Stereo Pipeline. The settings by which Vision Workbench processes is configurable by having a `.vwrc` file hidden in your home directory. Below is an example.

```
# This is an example VW configuration file. Save this file to ~/.vwrc
# to adjust the VW log settings, even if the program is already running.

# General settings
[general]
default_num_threads = 16
write_pool_size = 40
system_cache_size = 1024000000 # ~ 1 GB

# The following integers are associated with the log levels throughout the
# Vision Workbench.  Use these in the log rules below.
#
#    ErrorMessage = 0
#    WarningMessage = 10
#    InfoMessage = 20
#    DebugMessage = 30
#    VerboseDebugMessage = 40
#    EveryMessage = 100
#
# You can create a new log file or adjust the settings
# for the console log:
#   logfile <filename>
#       - or -
#   logfile console

# Once you have created a logfile (or selected the console), you can
# add log rules using the following syntax. (Note that you can use
# wildcard characters ’*’ to catch all log_levels for a given
# log_namespace, or vice versa.)

# <log_level> <log_namespace>

# Below are examples of using the log settings.

# Turn on various logging levels for several subsystems, with the
# output going to the console (standard output).
[logfile console]
# Turn on error and warning messages for the thread subsystem.
10 = thread
# Turn on error, warning, and info messages for the asp subsystem.
20 = asp
# Turn on error, warning, info, and debug messages for the stereo subsystem.
30 = stereo
# Turn on every single message for the cache subsystem (this will be
# extremely verbose and is not recommended).
# 100 = cache
# Turn off all progress bars to the console (not recommended).
# 0 = *.progress

# Turn on logging of error and warning messages to a file for the
# stereo subsystem. Warning: This file will be always appended to, so
# it should be deleted periodically.
# [logfile /tmp/vw_log.txt]
# 10 = stereo
```

There are a lot of possible options that can be implemented in the above example. Let’s cover the most important options and the concerns the user should have when selecting a value.

2.3.1 Performance Settings

default_num_threads (default=2)

This sets the maximum number of threads that can be used for rendering. When stereo's `subpixel_rfne` is running you'll probably notice 10 threads are running when you have `default_num_threads` set to 8. This is not an error, you are seeing 8 threads being used for rendering, 1 thread for holding `main()`'s execution, and finally 1 optional thread acting as the interface to the file driver.

It is usually best to set this parameter equal to the number of processors on your system. Be sure to include the number of logical processors in your arithmetic if your system supports hyper-threading.

Adding more threads for rasterization increases the memory demands of Stereo Pipeline. If your system is memory limited, it might be best to lower the `default_num_threads` option. Remember that 32 bit systems can only allocate 4 GB of memory per process. Despite Stereo Pipeline being a multithreaded application, it is still a single process.

write_pool_size (default=21)

The `write_pool_size` option represents the max waiting pool size of tiles waiting to be written to disk. Most file formats do not allow tiles to be written arbitrarily out of order. Most however will let rows of tiles to be written out of order, while tiles inside a row must be written in order. Because of the previous constraint, after a tile is rasterized it might spend some time waiting in the 'write pool' before it can be written to disk. If the 'write pool' fills up, only the next tile in order can be rasterized. That makes Stereo Pipeline perform like it is only using a single processor.

Increasing the `write_pool_size` makes Stereo Pipeline more able to use all processing cores in the system. Having this value too large can mean excessive use of memory. For 32 bit systems again, they can run out of memory if this value is too high for the same reason as described for `default_num_threads`.

system_cache_size (default=805306368)

Accessing a file from the hard drive can be very slow. It is especially bad if an application needs to make multiple passes over an input file. To increase performance, Vision Workbench will usually leave an input file stored in memory for quick access. This file storage is known as the 'system cache' and its max size is dictated by `system_cache_size`. The default value is 768 MB.

Setting this value too high can cause your application to crash. It is usually recommend to keep this value around 1/4 of the maximum available memory on the system. For 32 bit systems, this means don't set this value any greater than 1 GB. The units of this property is in bytes.

2.3.2 Logging Settings

The messages displayed in the console by Stereo Pipeline are grouped into several namespaces, and by level of verbosity. An example of customizing Stereo Pipeline's output is given in the `.vwrc` file shown above.

Several of the tools in Stereo Pipeline, including `stereo`, automatically append the information displayed in the console to a log file in the current output directory. These logs contain in addition some data about your system and settings, which may be helpful in resolving problems with the tools.

It is also possible to specify a global log file to which all tools will append to, as illustrated in `.vwrc`.

Chapter 3

Tutorial: Processing Mars Orbiter Camera Imagery

3.1 Quick Start

The Stereo Pipeline package contains command-line programs that convert a stereo pair in ISIS *cube* format into a 3D "point cloud" image: *stereo-output*-PC.tif. This is an intermediate format that can be passed along to one of several programs that convert a point cloud into a mesh for 3D viewing, a gridded digital elevation model (DEM) for GIS purposes, or a LAS/LAZ point cloud.

There are a number of ways to fine-tune parameters and analyze the results, but ultimately this software suite takes images and builds models in a mostly automatic way. To create a point cloud file, you simply pass two image files to the `stereo` command:

```
ISIS 3> stereo left_input_image.cub right_input_image.cub stereo-output
```

The string `stereo-output` is an arbitrary output prefix, it is used when generating names for `stereo` output files. For example, it can be set to `results/output`, in which case all output files will be in the `results` directory and start with the prefix `output`.

See section 5.1 for a more detailed discussion.

You can then make a visualizable mesh or a DEM file with the following commands (the *stereo-output*-PC.tif and *stereo-output*-L.tif files are created by the `stereo` program above):

```
ISIS 3> point2mesh stereo-output-PC.tif stereo-output-L.tif
ISIS 3> point2dem  stereo-output-PC.tif
```

More details are provided in section 5.2.

3.2 Preparing the Data

The data set that is used in the tutorial and examples below is a pair of Mars Orbital Camera (MOC) [18, 17] images whose Planetary Data System (PDS) Product IDs are M01/00115 and E02/01461. This data can be downloaded from the PDS directly, or they can be found in the `data/MOC/` directory of your Stereo Pipeline distribution.

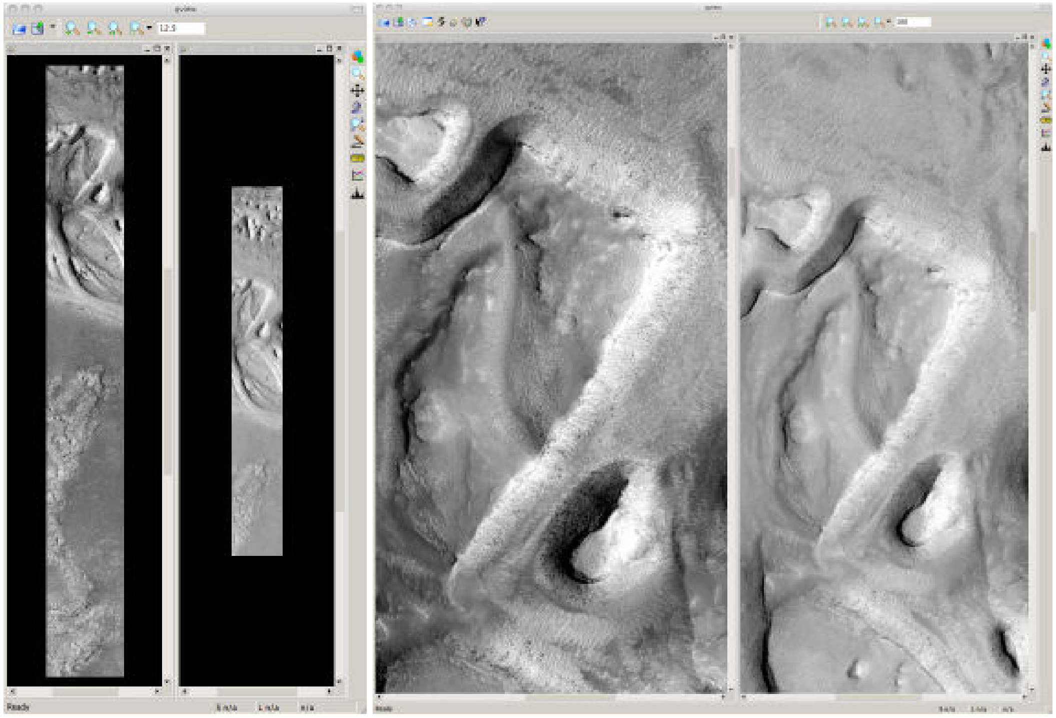


Figure 3.1: This figure shows E0201461.cub and M0100115.cub open in ISIS's qview program. The view on the left shows their full extents at the same zoom level, showing how they have different ground scales. The view on the right shows both images zoomed in on the same feature.

3.2.1 Loading and Calibrating Images using ISIS

These raw PDS images (`M0100115.imq` and `E0201461.imq`) need to be imported into the ISIS environment and radiometrically calibrated. You will need to be in an ISIS environment (have set the `ISISROOT` environment variable and sourced the appropriate ISIS 3 startup script, as detailed in the ISIS 3 instructions; we will denote this state with the '`ISIS 3>`' prompt). Then you can use the `mocproc` program, as follows:

```
ISIS 3> mocproc from=M0100115.imq to=M0100115.cub Mapping=NO
ISIS 3> mocproc from=E0201461.imq to=E0201461.cub Mapping=NO
```

There are also `Ingestion` and `Calibration` parameters whose defaults are '`YES`' which will bring the image into the ISIS format and perform radiometric calibration. By setting the `Mapping` parameter to '`NO`', the resultant file will be an ISIS cube file that is calibrated, but not map-projected. Note that while we have not explicitly run `spiceinit`, the Ingestion portion of `mocproc` quietly ran `spiceinit` for you (you'll find the record of it in the ISIS Session Log, usually written out to a file named `print.prt`). Refer to Figure 3.1 to see the results at this stage of processing.

Datasets for other type of cameras or other planets can be pre-processed similarly, using the ISIS tools specific to them.

3.2.2 Aligning Images

Once the .cub files are obtained, it is possible to run stereo right away, as

```
ISIS 3> stereo E0201461.cub M0100115.cub    \
          --alignment-method affineepipolar \
          -s stereo.default.example results/output
```

In this case, the first thing `stereo` does is to internally align (or rectify the images), which helps with finding stereo matches. Here we have used `affineepipolar` alignment. Another option is to use `homography` alignment, as described in section 5.1.1.

Alternatively, the images can be aligned externally, by map-projecting them in ISIS. External alignment can sometimes give better results than the simple internal alignment described earlier, especially if the images are taken from very different perspectives, or if the curvature of the planet/body being imaged is non-negligible. We will now describe how to do this alignment, but we also provide the `cam2map4stereo.py` program (page 92) which performs this work automatically for you.

The ISIS `cam2map` program will map-project these images:

```
ISIS 3> cam2map from=M0100115.cub to=M0100115.map.cub
ISIS 3> cam2map from=E0201461.cub to=E0201461.map.cub map=M0100115.map.cub matchmap=true
```

Notice the order in which the images were run through `cam2map`. The first projection with `M0100115.cub` produced a map-projected image centered on the center of that image. The projection of `E0201461.cub` used the `map=` parameter to indicate that `cam2map` should use the same map projection parameters as those of `M0100115.map.cub` (including center of projection, map extents, map scale, etc.) in creating the projected image. By map-projecting the image with the worse resolution first, and then matching to that, we ensure two things: (1) that the second image is summed or scaled down instead of being magnified up, and (2) that we are minimizing the file sizes to make processing in the Stereo Pipeline more efficient.

Technically, the same end result could be achieved by using the `mocproc` program alone, and using its `map= M0100115.map.cub` option for the run of `mocproc` on `E0201461.cub` (it behaves identically to `cam2map`). However, this would not allow for determining which of the two images had the worse resolution and extracting their minimum intersecting bounding box (see below). Furthermore, if you choose to conduct bundle adjustment (see Chapter 7, page 47) as a pre-processing step, you would do so between `mocproc` (as run above) and `cam2map`.

The above procedure is in the case of two images which cover similar real estate on the ground. If you have a pair of images where one image has a footprint on the ground that is much larger than the other, only the area that is common to both (the intersection of their areas) should be kept to perform correlation (since non-overlapping regions don't contribute to the stereo solution). If the image with the larger footprint size also happens to be the image with the better resolution (i.e. the image run through `cam2map` second with the `map=` parameter), then the above `cam2map` procedure with `matchmap=true` will take care of it just fine. Otherwise you'll need to figure out the latitude and longitude boundaries of the intersection boundary (with the ISIS `camrange` program). Then use that smaller boundary as the arguments to the `MINLAT`, `MAXLAT`, `MINLON`, and `MAXLON` parameters of the first run of `cam2map`. So in the above example, after `mocproc` with `Mapping= NO` you'd do this:

```
ISIS 3> camrange from=M0100115.cub
         ... lots of camrange output omitted ...
Group = UniversalGroundRange
  LatitudeType       = Planetocentric
  LongitudeDirection = PositiveEast
  LongitudeDomain    = 360
  MinimumLatitude    = 34.079818835324
  MaximumLatitude    = 34.436797628116
  MinimumLongitude   = 141.50666207418
  MaximumLongitude   = 141.62534719278
End_Group
         ... more output of camrange omitted ...
```

```
ISIS 3> camrange from=E0201461.cub
         ... lots of camrange output omitted ...
Group = UniversalGroundRange
  LatitudeType       = Planetocentric
  LongitudeDirection = PositiveEast
  LongitudeDomain    = 360
  MinimumLatitude    = 34.103893080982
  MaximumLatitude    = 34.547719435156
  MinimumLongitude   = 141.48853937384
  MaximumLongitude   = 141.62919740048
End_Group
         ... more output of camrange omitted ...
```

Now compare the boundaries of the two above and determine the intersection to use as the boundaries for `cam2map`:

```
ISIS 3> cam2map from=M0100115.cub to=M0100115.map.cub DEFAULTRANGE=CAMERA \
                MINLAT=34.10 MAXLAT=34.44 MINLON=141.50 MAXLON=141.63
ISIS 3> cam2map from=E0201461.cub to=E0201461.map.cub map=M0100115.map.cub matchmap=true
```

You only have to do the boundaries explicitly for the first run of `cam2map`, because the second one uses the `map=` parameter to mimic the map-projection of the first. These two images are not radically different in areal coverage, so this is not really necessary for these images, it is just an example.

Again, unless you are doing something complicated, using the `cam2map4stereo.py` program (page 92) will take care of all these steps for you.

At this stage we can run the stereo program with map-projected images:

```
ISIS 3> stereo E0201461.map.cub M0100115.map.cub --alignment-method none \
          -s stereo.default.example results/output
```

Now you may skip to chapter 5 which will discuss the `stereo` program in more detail and the other tools in ASP.

Chapter 4

Tutorial: Processing Earth Digital Globe Imagery

In this chapter we will focus on how to process Earth imagery, or more specifically Digital Globe imagery. This is different from our previous chapter in that at no point will we be using ISIS utilities. This is because ISIS only supports NASA instruments, while most Earth imagery comes from commercial providers.

Digital Globe provides imagery from Quick Bird and the two World View satellites. These are the hardest images to process with Ames Stereo Pipeline because they are exceedingly large, much larger than HiRISE imagery. There is also a wide range of terrain challenges and atmospheric effects that can confuse ASP. Trees are particularly difficult for us since their texture is nearly nadir and perpendicular to our line of sight. It is important to know that the driving force behind our support for Digital Globe imagery is to create models of ice and bare rock. That is the type of imagery that we have tested with and have focused on. If we can make models of wooded or urban areas, that is a bonus, but we can't provide any advice for how to perform or improve the results if you choose to use ASP in that way.

ASP can only process Level 1B satellite imagery, and cannot process Digital Globe's aerial images.

The camera information for Digital Globe images is contained in an XML file for each image. In addition to the exact linear camera model, the XML file also has its RPC approximation. In this chapter we will focus only on processing data using the linear camera model. For more detail on RPC camera models we refer to section 8.10 on page 68, which discusses processing GeoEye imagery which comes only with RPC coefficients.

Our implementation of the linear camera model only models the geometry of the imaging hardware itself and velocity aberration. We do not currently model refraction due to light bending in Earth's atmosphere. It is our understanding that this could represent misplacement of points up to a meter for some imagery. However this is still smaller error than the error from measurement of the spacecraft's position and orientation. We do not provide facilities for correcting spacecraft altitude either. However, the `pc_align` tool discussed in section 5.2.4 can be used to align the terrain obtained from Stereo Pipeline to an accurate set of ground measurements.

In the next two sections we will show how to process unmodified and map-projected variants of World View imagery. This steps will be the same for Digital Globe's other satellites. The imagery we are using are from the free stereo pair example of Lucknow, India available from Digital Globe's website [13]. These images represent a non-ideal problem for us since this is an urban location, but at least you should be able to download this imagery yourself and follow along.

4.1 Processing Raw

After you have downloaded the example stereo imagery of India, you will find a directory titled 052783824050_01_P001_PAN. It has a lot of files and many of them contain redundant information just displayed in different formats. We are interested only in the TIF or NTF imagery and the similarly named XML files.

Further investigation of the files downloaded will show that there are in fact 4 image files. This is because Digital Globe breaks down a single observation into multiple files for what we assume are size reasons. These files have a pattern string of "_R[N]C1-", where N increments for every subframe of the full observation. The tool named `dg_mosaic` can be used to mosaic (and optionally reduce the resolution of) such a set of sub-observations into a single image file and create an appropriate camera file

```
> dg_mosaic 12FEB12053305*TIF --output-prefix 12FEB12053305 --reduce-percent 50
```

and analogously for the second set. See section A.8 for more details. The `stereo` program can use either the original or the mosaicked images.

Since we are ingesting these images raw, it is strongly recommended that you use affine epipolar alignment to reduce the search range. The `stereo` command and a rendering in QGIS are shown below.

```
> stereo -t dg --subpixel-mode 1 --alignment-method affineepipolar \
        12FEB12053305-P1BS_R2C1-052783824050_01_P001.TIF          \
        12FEB12053341-P1BS_R2C1-052783824050_01_P001.TIF          \
        12FEB12053305-P1BS_R2C1-052783824050_01_P001.XML          \
        12FEB12053341-P1BS_R2C1-052783824050_01_P001.XML dg/dg
```

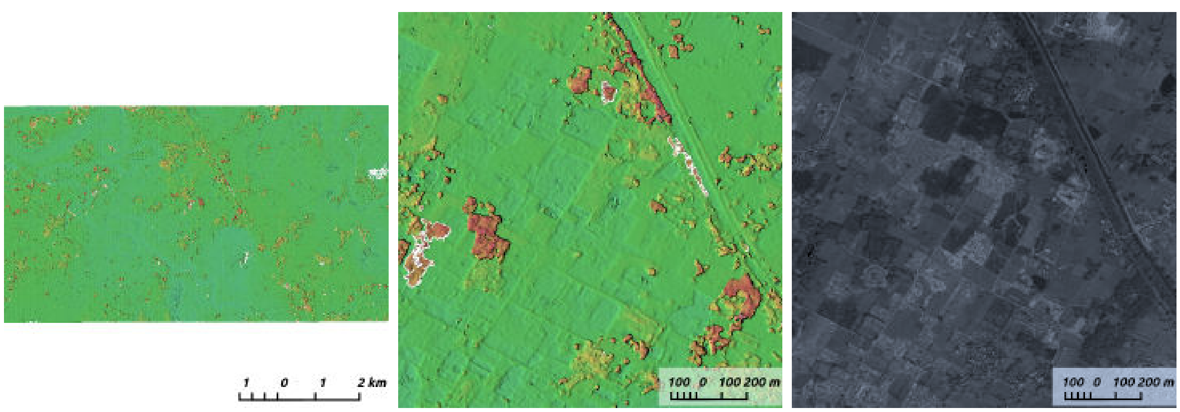


Figure 4.1: Example colorized height map and ortho image output.

Above, we have used `subpixel-mode 1` which is less accurate but reasonably fast. More details about how to set this and other `stereo` parameters can be found in section 5.1.1.

It is important to note that we could have performed stereo using the approximate RPC model instead of the exact linear camera model (both models are in the same XML file), by switching the session in the `stereo` command above from `-t dg` to `-t rpc`. The RPC model is somewhat less accurate, so the results will not be the same, in our experiments we've seen differences in the 3D terrains using the two approaches of 5 meters or more.

4.2 Processing Map-Projected Imagery

Eventually you will run into Digital Globe imagery that has too much parallax to be processed in a reasonable time. (That was not the case for Lucknow, India because it is so flat.) We can speed up the result by performing stereo on map-projected versions of the images. The map-projection is done with a tool named `mapproject` (section A.9). It uses the simplified RPC model contained in the camera XML file to project a given camera image onto a pre-determined low-resolution DEM without holes.

ASP will then perform correlation on the map-projected images, and, before doing triangulation will internally project back the image pixels onto the original camera locations, precisely reversing the transformation done with `mapproject`. Since map-projection is a temporary pre-processing step, it is of little importance that the DEM is low-resolution, or that we use the less accurate (but faster) RPC model to perform it.

The hardest part of this whole process is getting the input low-resolution DEM. In this example we will use a variant of NASA SRTM data with no holes. Other choices might be GMTED2010, USGS's NED data, or NGA's DTED data.

It is important to note that ASP expects the input low-resolution DEM to be in reference to a datum ellipsoid, such as WGS84 or NAD83. If the DEM is in respect to either the EGM96 or NAVD88 geoids, the ASP tool `dem_geoid` can be used to convert the DEM to WGS84 or NAD83 (section A.7). (The same tool can be used to convert back the final output ASP DEM to be in reference to a geoid, if desired.)

Not applying this conversion might not properly negate the parallax seen between the two images, though it will not corrupt the triangulation results. In other words, sometimes one may be able to ignore the vertical datums on the input but we do not recommend doing that. Also, you should note that the geoheader attached to those types of files usually does not describe the vertical datum they used. That can only be understood by careful reading of your provider's documents.

In this example we use as an input low-resolution DEM the file `srtm_53_07.tif`, a 90 meter resolution tile from the CGIAR-CSI modification of the original NASA SRTM product [9]. The NASA SRTM square for this example spot in India is N26E080.

Below are the commands for map-projecting the input and then running through stereo. You can use any projection you like as long as it preserves detail in the imagery. Note that the last parameter in the stereo call is the input low-resolution DEM.

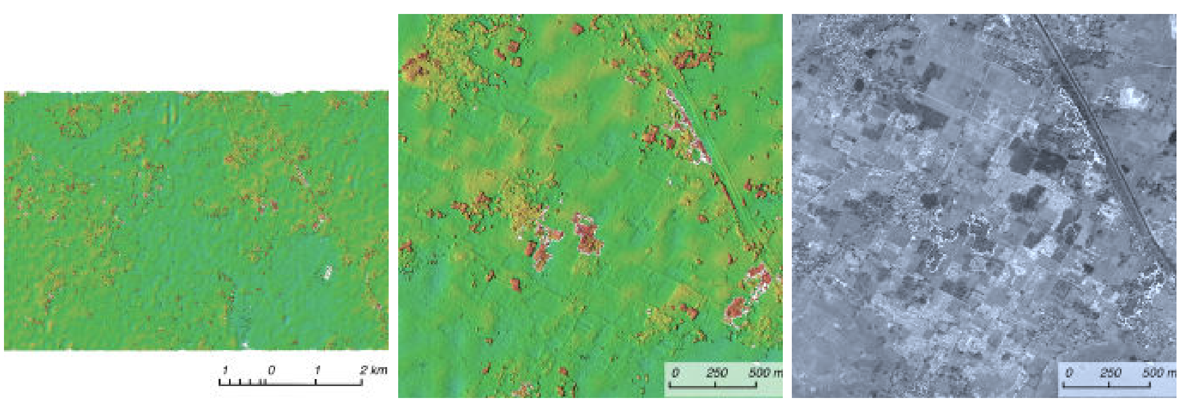


Figure 4.2: Example colorized height map and ortho image output.

Commands

```
> mapproject --t_srs "+proj=eqc +units=m +datum=WGS84" \
```

```
    --tr 0.5 srtm_53_07.tif                             \
    12FEB12053305-P1BS_R2C1-052783824050_01_P001.TIF   \
    12FEB12053305-P1BS_R2C1-052783824050_01_P001.XML   \
    left_mapped.tif
> mapproject --t_srs "+proj=eqc +units=m +datum=WGS84" \
    --tr 0.5 srtm_53_07.tif                             \
    12FEB12053341-P1BS_R2C1-052783824050_01_P001.TIF   \
    12FEB12053341-P1BS_R2C1-052783824050_01_P001.XML   \
    right_mapped.tif
> stereo -t dg --subpixel-mode 1 --alignment-method none  \
         left_mapped.tif right_mapped.tif                 \
         12FEB12053305-P1BS_R2C1-052783824050_01_P001.XML \
         12FEB12053341-P1BS_R2C1-052783824050_01_P001.XML \
         dg/dg srtm_53_07.tif
```

If the `-t_srs` option is not specified, it will be read from the low-resolution input DEM.

The complete list of options for `mapproject` is described in section A.9.

In the `stereo` command, we have used `subpixel-mode 1` which is less accurate but reasonably fast. We have also used `alignment-method none`, since the images are map-projected, and thus no alignment is necessary. More details about how to set these and other `stereo` parameters can be found in section 5.1.1.

4.3 Handling CCD Boundary Artifacts

Digital Globe World View images [12] may exhibit slight subpixel artifacts which manifest themselves as discontinuities in the 3D terrain obtained using ASP. We provide a tool named `wv_correct` which can largely correct such artifacts for World View-1 and World View-2 images for most TDI. It can be invoked as follows:

```
> wv_correct image.ntf image.xml image_out.tif
```

The corrected images can be used just as the originals, and the camera models do not change. So one can mosaic them, perform map-projection, do stereo, etc.

This tool is described in section A.15, and an example of using it is in Figure 4.3.

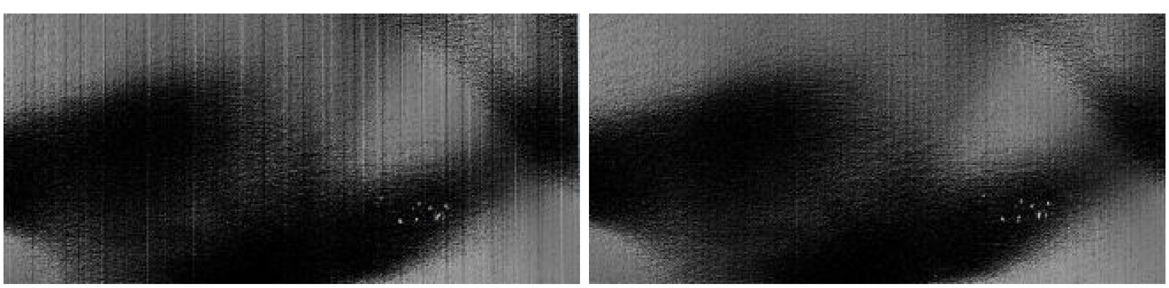


Figure 4.3: Example of a hill-shaded terrain obtained using stereo without (left) and with (right) CCD boundary artifact corrections applied using `wv_correct`.

4.4 Dealing with Terrain Lacking Large Scale Features

Stereo Pipeline's approach to performing correlation is a two-step pyramid algorithm, in which low-resolution versions of the input images are created, the disparity map (*output_prefix*-D_sub.tif) is found, and then this disparity map is refined using increasingly higher-resolution versions of the input images (section 6.2).

This approach usually works quite well for rocky terrain but may fail for snowy landscapes, whose only features may be small-scale grooves or ridges sculpted by wind (so-called *zastrugi*) that disappear at low resolution.

Stereo Pipeline handles such terrains by using a tool named `sparse_disp` to create *output_prefix*-D_sub.tif at full resolution, yet only at a sparse set of pixels for reasons of speed. This low-resolution disparity is then refined as earlier using a pyramid approach.

Figure 4.4: Example of a difficult terrain obtained without (left) and with (right) `sparse_disp`. (In these DEMs there is very little elevation change, hence the flat appearance.)

This mode can be invoked by passing to `stereo` the option `--corr-seed-mode 3`. Also, during pyramid correlation it is suggested to use somewhat fewer levels than the default `--corr-max-levels 5`, to again not subsample the images too much and lose the features.

Here is an example:

```
> stereo -t dg --corr-seed-mode 3 --corr-max-levels 2     \
         left_mapped.tif right_mapped.tif                 \
         12FEB12053305-P1BS_R2C1-052783824050_01_P001.XML \
         12FEB12053341-P1BS_R2C1-052783824050_01_P001.XML \
         dg/dg srtm_53_07.tif
```

It is important to note that `sparse_disp` is written in Python and depends on a variety of binary Python modules. These modules cannot be distributed with Stereo Pipeline as they depend on the version of Python installed on your system.

We provide a script which will download and compile the dependencies of this tool for your platform. The script and instructions are at https://github.com/NeoGeographyToolkit/BinaryBuilder/tree/master/build_python_modules

4.5 Processing Multi-Spectral Images

In addition to panchromatic (grayscale) imagery, the Digital Globe satellites also produce lower-resolution multi-spectral (multi-band) images. Stereo Pipeline is designed to process single-band images only. If invoked on multi-spectral data, it will quietly process the first band and ignore the rest. To use one of the other bands it can be singled out by invoking `dg_mosaic` (section 4.1) with the `--band <num>` option. We have evaluated ASP with Digital Globe's multi-spectral images, but support for it is still experimental. We recommend using the panchromatic imagery whenever possible.

Chapter 5

The Next Steps

This chapter will discuss in more detail ASP's stereo process and other tools available to either pre-process the input images/cameras or to manipulate `stereo`'s outputs, both in the context of planetary ISIS data and for Earth imagery. This includes how to (a) customize `stereo`'s settings (b) use `point2dem` to create 3D terrain models, (c) visualize the results, (d) align the obtained point clouds to another data source, (e) perform 3D terrain adjustments in respect to a geoid, etc.

5.1 Stereo Pipeline in More Detail

5.1.1 Setting Options in the `stereo.default` File

The `stereo` program requires a `stereo.default` file that contains settings that affect the stereo reconstruction process. Its contents can be altered for your needs; details are found in appendix B on page 99. You may find it useful to save multiple versions of the `stereo.default` file for various processing needs. If you do this, be sure to specify the desired settings file by invoking `stereo` with the `-s` option. If this option is not given, the `stereo` program will search for a file named `stereo.default` in the current working directory. If `stereo` does not find `stereo.default` in the current working directory and no file was given with the `-s` option, `stereo` will assume default settings and continue.

An example `stereo.default` file is available in the `examples/` directory of ASP. The actual file has a lot of comments to show you what options and values are possible. Here's a trimmed version of the important values in that file.

```
alignment-method affineepipolar
cost-mode 2
corr-kernel 21 21
subpixel-mode 1
subpixel-kernel 21 21
```

All these options can be overridden from the command line, as described in section 5.1.3.

Alignment Method

The most important line in `stereo.default` is the first one, specifying the alignment method. For raw images, alignment is always necessary, as the left and right images are from different perspectives. Several alignment methods are supported, including `affineepipolar` and `homography` (see section B.1 for details).

Alternatively, stereo can be performed with map-projected images (section 3.2.2 for ISIS and section 4.2 for Earth imagery). In effect we take a smooth low-resolution terrain and map both the left and right raw images onto that terrain. This automatically brings both images into the same perspective, and as such, for map-projected images the alignment method is always set to `none`.

Correlation Parameters

The second and third lines in `stereo.default` define what correlation metric *(normalized cross correlation)* we'll be using and how big the template or kernel size should be *(21 pixels square)*. A pixel in the left image will be matched to a pixel in the right image by comparing the windows of this size centered at them.

Making the kernel sizes smaller, such as 15×15, or even 11×11, may improve results on more complex features, such as steep cliffs, at the expense of perhaps introducing more false matches or noise.

Subpixel Refinement Parameters

A highly critical parameter in ASP is the value of `subpixel-mode`, on the fourth line. When set to 1, `stereo` performs parabola subpixel refinement, which is very fast but not very accurate. When set to 2, it produces very accurate results, but it is about an order of magnitude slower. When set to 3, the accuracy and speed will be somewhere in between the other methods.

The fifth line sets the kernel size to use during subpixel refinement *(also 21 pixels square)*.

Search Range Determination

Using these settings alone, ASP will attempt to work out the minimum and maximum disparity it will search for automatically. However if you wish to, you can explicitly set the extent of the search range by adding the option:

```
corr-search -80 -2 20 2
```

The exact values to use with this option you'll have to discover yourself. The numbers right of `corr-search` represent the horizontal minimum boundary, vertical minimum boundary, horizontal maximum boundary, and finally the horizontal maximum boundary.

It can be tricky to select a good search range for the `stereo.default` file. That's why the best way is to let `stereo` perform an automated guess for the search range. If you find that you can do a better estimate of the search range, take look at the intermediate disparity images using the `disparitydebug` program to figure out which search directions can be expanded or contracted. The output images will clearly show good data or bad data depending on whether the search range is correct.

The worst case scenario is to determine the search range manually. For example, for ISIS images, both images could be opened in `qview` and the coordinates of points that can be matched visually can be compared. Subtract line,sample locations in the first image from the coordinates of the same feature in the second image, and this will yield offsets that can be used in the search range. Make several of these offset measurements and use them to define a line,sample bounding box, then expand this by 50% and use it for `corr-search`. This will produce good results in most images.

Also, if you are using an alignment option, you'll instead want to make those disparity measurements against the written L.tif and R.tif files (see chapter C) instead of the original input files.

Figure 5.1: These are the four viewable `.tif` files created by the `stereo` program. On the left are the two aligned, pre-processed images: (`results/output-L.tif` and `results/output-R.tif`). The next two are mask images (`results/output-lMask.tif` and `results/output-rMask.tif`), which indicate which pixels in the aligned images are good to use in stereo correlation. The image on the right is the "Good Pixel map", (`results/output-GoodPixelMap.tif`), which indicates (in gray) which were successfully matched with the correlator, and (in red) those that were not matched.

5.1.2 Performing Stereo Correlation

As already mentioned, the `stereo` program can be invoked for ISIS images as

```
ISIS 3> stereo left_image.cub right_image.cub \
          -s stereo.default results/output
```

For Digital Globe imagery the cameras need to be specified separately:

```
> stereo left.tif right.tif left.xml right.xml \
    -s stereo.default results/output
```

As stated in section 3.1, the string `results/output` is arbitrary, and in this case we will simply make all outputs go to the `results` directory.

When `stereo` finishes, it will have produced a point cloud image. Section 5.2 describes how to convert it to a digital elevation model (DEM) or other formats.

The `stereo` command can also take multiple input images, performing multi-view stereo (section 5.1.5).

5.1.3 Specifying Settings on the Command Line

All the settings given via the `stereo.default` file can be over-ridden from the command line. Just add a double hyphen (`--`) in front the option's name and then fill out the option just as you would in the configuration file. For options in the `stereo.default` file that take multiple numbers, they must be separated by spaces (like '`corr-kernel 25 25`') on the command line. Here is an example in which we override the search range and subpixel mode from the command line.

```
ISIS 3> stereo E0201461.map.cub M0100115.map.cub  \
          -s stereo.map --corr-search -70 -4 40 4 \
          --subpixel-mode 0 results/output
```

5.1.4 Stereo on Multiple Machines

If the input images are really large it may desirable to distribute the work over several computing nodes. ASP provides a tool named `parallel_stereo` for that purpose. Its usage is described in section A.2.

5.1.5 Multi-View Stereo

ASP does not support full multi-view stereo in which a feature is correlated simultaneously across multiple images. However, it does offer several approaches of combining the results of processing multiple images into one single terrain model. In any of the scenarios described below, it may be helpful to first bundle-adjust the images (section 7.2).

Given several images, the most basic method is to perform pairwise stereo and generate one point cloud and one DEM for each pair (section 5.2.2), then combine the DEMs with `dem_mosaic` (section A.6).

Alternatively, multiple point clouds can be created, and then a single DEM can be created with `point2dem` (section A.4). In both of these approaches, the point clouds could be first registered to a trusted dataset using `pc_align` before creating terrain models (section 5.2.4).

Lastly, the `stereo` command can be invoked with multiple images. The rest of this section describes this approach. In this scenario, the first image is set as reference, then disparities are computed from it to the other ones, and then joint triangulation is performed using the method from [26]. A single point cloud is generated with one 3D point for each pixel in the first image. The inputs to multi-view stereo and its output point cloud can be handled in the same way as for two-view stereo (e.g., inputs can be map-projected, the output can be converted to a DEM, etc.).

Example (for ISIS with three images):

```
stereo file1.cub file2.cub file3.cub results/run
```

Example (for Digital Globe data with three map-projected images):

```
stereo file1.tif file2.tif file3.tif file1.xml file2.xml file3.xml \
  results/run input-DEM.tif
```

The `parallel_stereo` tool can also be used with multiple images (section A.2).

If the images are in a sequence, one of the images in the middle of the sequence can be used as a reference (so it needs to be passed to the tools as the first image).

The ray intersection error, the fourth band in the point cloud file, is computed as twice the mean of distances from the optimally computed intersection point to the individual rays. For two rays, this agrees with the intersection error for two-view stereo which is defined as the minimal distance between rays. For multi-view stereo this error is much less amenable to interpretation as for two-view stereo, since the number of valid rays corresponding to a given feature can vary across the image, which results in discontinuities in the intersection error.

5.1.6 Diagnosing Problems

Once invoked, `stereo` proceeds through several stages that are detailed on page 74. Intermediate and final output files are generated as it goes. See Appendix C, page 105 for a comprehensive listing. Many of these files are useful for diagnosing and debugging problems. For example, as Figure 5.1 shows, a quick look at some of the TIFF files in the `results/` directory provides some insight into the process.

Perhaps the most accessible file for assessing the quality of your results is the good pixel image, (`results/output-GoodPixelMap.tif`). If this file shows mostly good, gray pixels in the overlap area (the area that is white in both the `results/output-lMask.tif` and `results/output-rMask.tif` files), then your results are just fine. If the good pixel image shows lots of failed data, signified by red pixels in the overlap area, then you need to go back and tune your `stereo.default` file until your results improve. This might be a good time to make a copy of `stereo.default` as you tune the parameters to improve the results.

You should also know that whenever `stereo`, `point2dem`, and other executables are run, they create log files in given tool's results directory, containing a copy of the configuration file, the command that was run, your system settings, and tool's console output. This will help track what was performed so that others in the future can recreate your work.

Another handy debugging tool is the `disparitydebug` program, which allows you to generate viewable versions of the intermediate results from the stereo correlation algorithm. `disparitydebug` converts information in the disparity image files into two TIFF images that contain horizontal and vertical components of the disparity (i.e. matching offsets for each pixel in the horizontal and vertical directions). There are actually three flavors of disparity map: the `-D.tif`, the `-RD.tif`, and `-F.tif`. You can run `disparitydebug` on any of them. Each shows the disparity map at the different stages of processing.

```
> disparitydebug results/output-F.tif
```

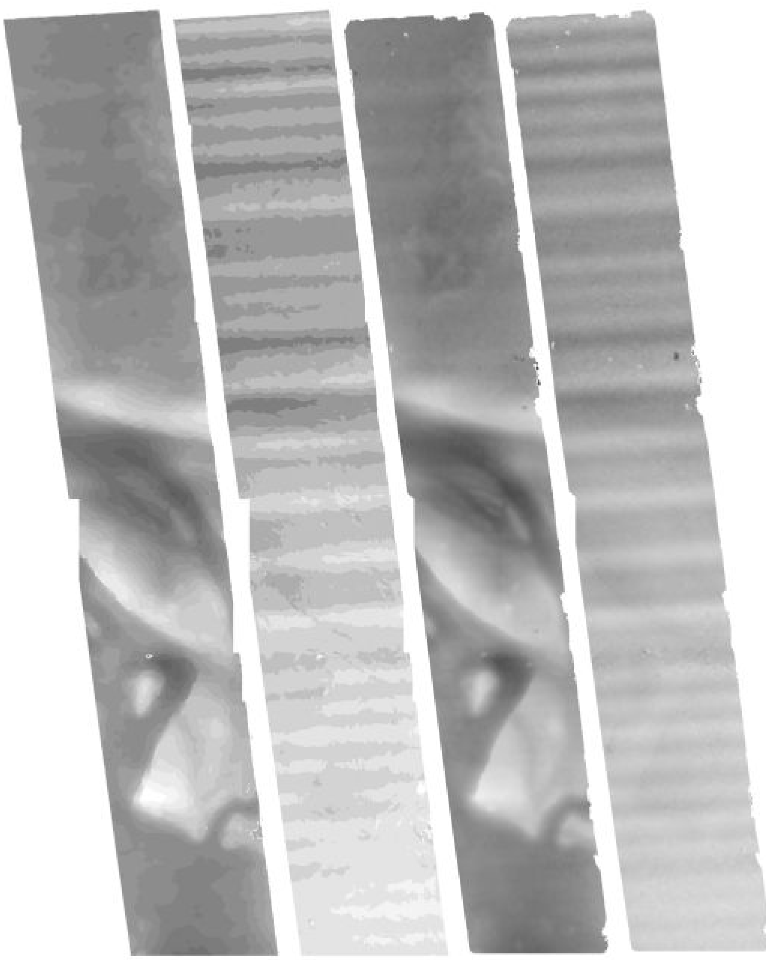


Figure 5.2: Disparity images produced using the `disparitydebug` tool. The two images on the left are the `results/output-D-H.tif` and `results/output-D-V.tif` files, which are normalized horizontal and vertical disparity components produced by the disparity map initialization phase. The two images on the right are `results/output-F-H.tif` and `results/output-F-V.tif`, which are the final filtered, sub-pixel-refined disparity maps that are fed into the Triangulation phase to build the point cloud image. Since these MOC images were acquired by rolling the spacecraft across-track, most of the disparity that represents topography is present in the horizontal disparity map. The vertical disparity map shows disparity due to "washboarding," which is not from topography but from spacecraft movement. Note however that the horizontal and vertical disparity images are normalized independently. Although both have the same range of gray values from white to black, they represent significantly different absolute ranges of disparity.

If the output H and V files from `disparitydebug` look good, then the point cloud image is most likely ready for post-processing. You can proceed to make a mesh or a DEM by processing `results/output-PC.tif` using the `point2mesh` or `point2dem` tools, respectively.

Figure 5.2 shows the outputs of `disparitydebug`.

And a note on performance. If `stereo_corr` takes unreasonably long, it may have encountered a portion of the image where, due to noise (such as clouds, shadows, etc.) the determined search range is much larger than what it should be. The option `--corr-timeout` *`integer`* can be used to limit how long each 1024×1024 pixel tile can take. A good value here could be 300 (seconds) or more if your terrain is expected to have large height variations.

5.2 Visualizing and Manipulating the Results

When `stereo` finishes, it will have produced a point cloud image. At this point, many kinds of data products can be built from the `results/output-PC.tif` point cloud file.

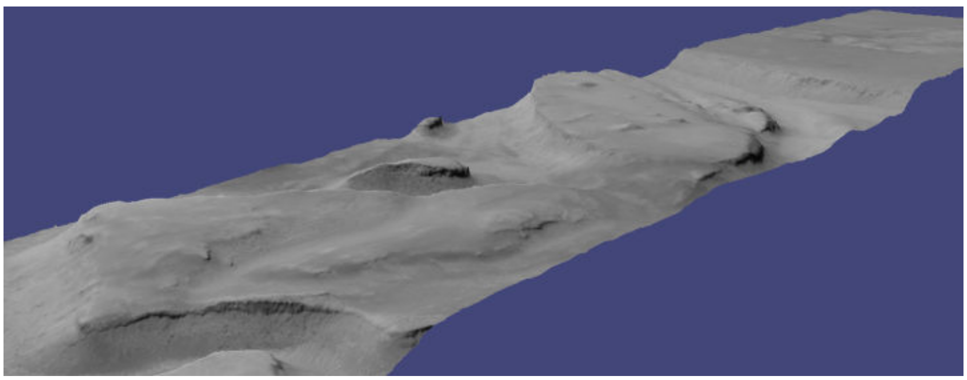


Figure 5.3: The `results/output.osgb` file displayed in the OSG Viewer.

5.2.1 Building a 3D Model

If you wish to see the data in an interactive 3D browser, then you can generate a 3D object file using the `point2mesh` command (page 83). The resulting file is stored in Open Scene Graph binary format [8]. It can be viewed with `osgviewer` (the Open Scene Graph Viewer program, distributed with the binary version of the Stereo Pipeline). The `point2mesh` program takes the point cloud file and the left normalized image as inputs:

```
> point2mesh results/output-PC.tif results/output-L.tif
> osgviewer results/output.osgb
```

The image displayed by `osgviewer` is shown in figure 5.3.

When the `osgviewer` program starts, you may want to toggle the lighting with the 'L' key, toggle texturing with the 'T' key, and toggle wireframe mode with the 'W'. Press '?' to see a variety of other interactive options.

5.2.2 Building a Digital Elevation Model

The `point2dem` program (page 79) creates a Digital Elevation Model (DEM) from the point cloud file.

```
> point2dem results/output-PC.tif
```

The resulting TIFF file is map-projected and will contain georeferencing information stored as GeoTIFF tags. You can specify a coordinate system (e.g., mercator, sinusoidal) and a reference spheroid (i.e., calculated for the Moon, Mars, or Earth).

```
> point2dem -r mars results/output-PC.tif
```

This product is suitable for scientific use, and can be imported into a variety of GIS platforms. However, the resulting file, `results/output-DEM.tif`, will have 32-bit floating point pixels, and will not render well in typical image viewers.

The `point2dem` program can also be used to orthoproject raw satellite imagery onto the DEM. To do this, invoke `point2dem` just as before, but add the `--orthoimage` option and specify the use of the left image file as the texture file to use for the projection:

```
> point2dem -r mars  results/output-PC.tif --orthoimage results/output-L.tif
```

The texture file must always be specified after the point cloud file. See figure 5.4 on the right for the output of this command.

If the DEM has holes, which can be inevitable, those holes will also show up in the orthoimage. They can be filled in using the option `--orthoimage-hole-fill-len` with a value passed to it.

The `point2dem` program is also able to accept output projection options the same way as the tools in GDAL. Well-known EPSG, IAU2000 projections, and custom Proj4 strings can applied with the target spatial reference set flag, `--t_srs`. If the target spatial reference flag is applied with any of the reference spheroid options, the reference spheroid option will overwrite the datum defined in the target spatial reference set. The following examples produce the same output.

```
> point2dem --t_srs IAU2000:49900 results/output-PC.tif
> point2dem --t_srs "+proj=longlat +a=3396190 +b=3376200"
    results/output-PC.tif
```

The `point2dem` program can be used in many different ways. The complete documentation is in section A.4.

5.2.3 Fine-Tuning the Results

There are several options in Stereo Pipeline that, when adjusted, can help produce higher quality output.

During the filtering step of stereo (section A.1.2), one can choose between several ways of removing outliers, control how much hole-filling should take place, if at all, and if to remove small isolated regions from the output. This is detailed in section B.4. In the latest iterations of ASP, we suggest that at this stage only the removal of small regions should take place, while outlier removal and hole-filling be delayed until triangulation and 3D terrain generation, as described in the next paragraph.

During the triangulation step, erroneous points in the output point cloud can be filtered out based on a range of distances from either the left camera or the planet center. More details are in section B.5. When creating a 3D terrain model with `point2dem`, additional customizable outlier removal based on triangulation error takes place, and small holes can be filled in the DEM itself or the orthoimage. Lastly, boundary DEM pixels (which are sometimes noisy) can be eroded using the `dem_mosaic` tool (section A.6) whose main purpose is to mosaic DEMs.

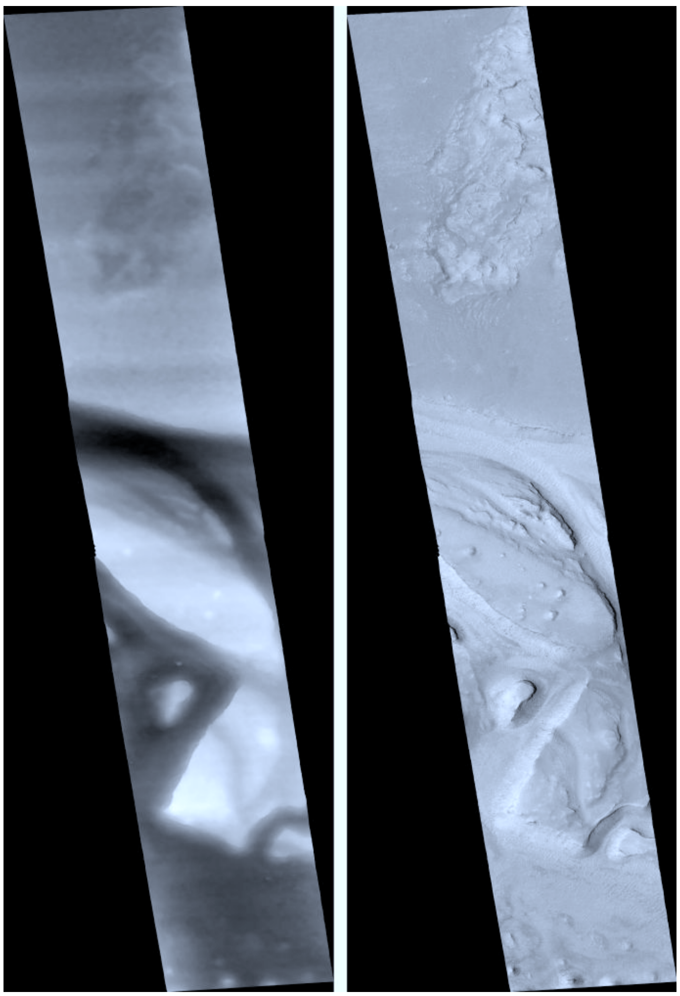


Figure 5.4: The image on the left is a normalized DEM (generated using `point2dem`'s `-n` option), which shows low terrain values as black and high terrain values as white. The image on the right is the left input image projected onto the DEM (created using the `--orthoimage` option to `point2dem`).

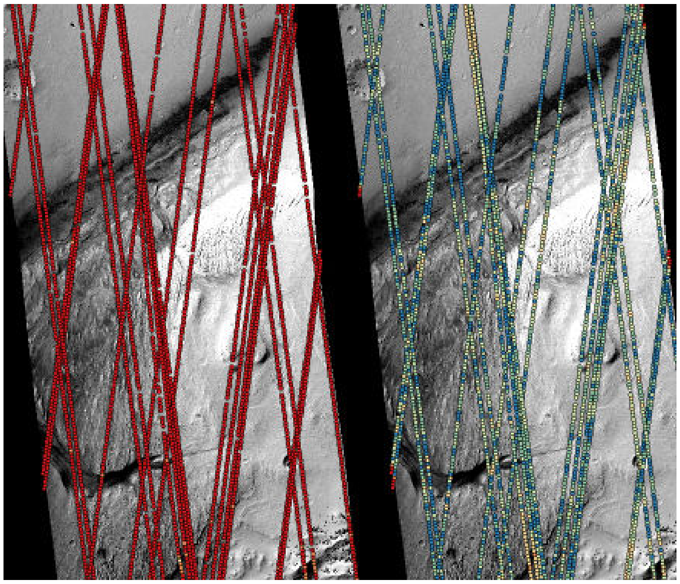


Figure 5.5: Example of using `pc_align` to align a DEM obtained using stereo from CTX images to a set of MOLA tracks. The MOLA points are colored by the offset error initially (left) and after pc align was applied (right) to the terrain model. The red dots indicate more than 100 m of error and blue less than 5 m. The `pc_align` algorithm determined that by moving the terrain model approximately 40 m south, 70 m west, and 175 m vertically, goodness of fit between MOLA and the CTX model was increased substantially.

5.2.4 Alignment to Point Clouds From a Different Source

Often the 3D terrain models output by `stereo` (point clouds and DEMs) can be intrinsically quite accurate yet their actual position on the planet may be off by several meters or several kilometers, depending on the spacecraft. This can result from small errors in the position and orientation of the satellite cameras taking the pictures.

ASP provides a tool named `pc_align` for aligning such 3D terrains to a much more accurately positioned (if potentially sparser) dataset. Such datasets can be made up of ground control points (in the case of Earth), or from laser altimetry instruments on satellites, such as ICESat/GLASS for Earth, LRO/LOLA on the Moon, and MGS/MOLA on Mars. Under the hood, `pc_align` uses the Iterative Closest Point algorithm (ICP) (both the point-to-plane and point-to-point flavors are supported).

The `pc_align` tool requires another input, an a priori guess for the maximum displacement we expect to see as result of alignment, i.e., by how much the points are allowed to move when the alignment transform is applied. If not known, a large (but not unreasonably so) number can be specified. It is used to remove most of the points in the source (movable) point cloud which have no chance of having a corresponding point in the reference (fixed) point cloud.

Here is how `pc_align` can be called (the denser cloud is specified first).

```
> pc_align --max-displacement 200 --datum D_MARS \
    --save-inv-transformed-reference-points      \
    --csv-format '1:lon 2:lat 3:radius_m'      \
    stereo-PC.tif mola.csv
```

Figure 5.5 shows an example of using `pc_align`. The complete documentation for this program is in section A.14.

5.2.5 Creating DEMs Relative to the Geoid/Areoid

The DEMs generated using `point2dem` are in reference to a datum ellipsoid. If desired, the `dem_geoid` program can be used to convert this DEM to be relative to a geoid/areoid on Earth/Mars respectively. Example usage:

```
> dem_geoid results/output-DEM.tif
```

5.2.6 Converting to the LAS Format

If it is desired to use the `stereo` generated point cloud outside of ASP, it can be converted to the LAS file format, which is a public file format for the interchange of 3-dimensional point cloud data. The tool `point2las` can be used for that purpose (section A.13). Example usage:

```
> point2las --compressed -r Earth results/output-PC.tif
```

5.2.7 Generating Color Hillshade Maps

Once you have generated a DEM file, you can use the `colormap` and `hillshade` tools to create colorized and/or shaded relief images.

To create a colorized version of the DEM, you need only specify the DEM file to use. The colormap is applied to the full range of the DEM, which is computed automatically. Alternatively you can specify your own min and max range for the color map.

```
> colormap results/output-DEM.tif -o hrad-colorized.tif
```

To create a hillshade of the DEM, specify the DEM file to use. You can control the azimuth and elevation of the light source using the `-a` and `-e` options.

```
> hillshade results/output-DEM.tif -o hrad-shaded.tif -e 25
```

To create a colorized version of the shaded relief file, specify the DEM and the shaded relief file that should be used:

```
> colormap results/output-DEM.tif -s hrad-shaded.tif -o hrad-color-shaded.tif
```

See figure 5.6 showing the images obtained with these commands.

5.2.8 Building Overlays for Moon and Mars Mode in Google Earth

The final program in the Stereo Pipeline package that this tutorial will address is `image2qtree`. This tool was designed to create tiled, multi-resolution overlays for Google Earth. In addition to generating image tiles, it produces a metadata tree in KML format that can be loaded from your local hard drive or streamed from a remote server over the Internet.

The `image2qtree` program can only be used on 8-bit image files with georeferencing information (e.g. grayscale or RGB GeoTIFF images). In this example, it can be used to process `results/output-DEM-normalized.tif`, `results/output-DRG.tif`, `hrad-shaded.tif`, `hrad-colorized.tif`, and `hrad-shaded-colorized.tif`.

These images were generated respectively by using `point2dem` with the `-n` option creating a normalized DEM, the `--orthoimage` option to `point2dem` which projects the left image onto the DEM, and the images created earlier with `colormap`.

```
> image2qtree hrad-shaded-colorized.tif -m kml --draw-order 100
```

Figure 5.7 shows the obtained KML files in Google Earth.

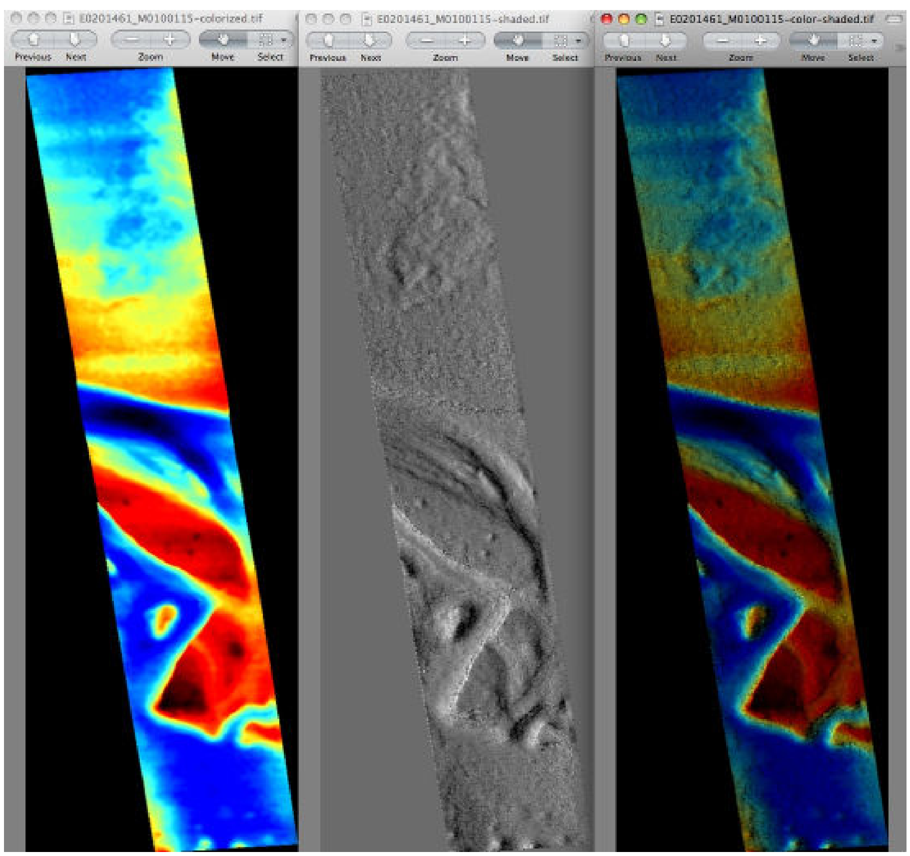


Figure 5.6: The colorized DEM, the shaded relief image, and the colorized hillshade.

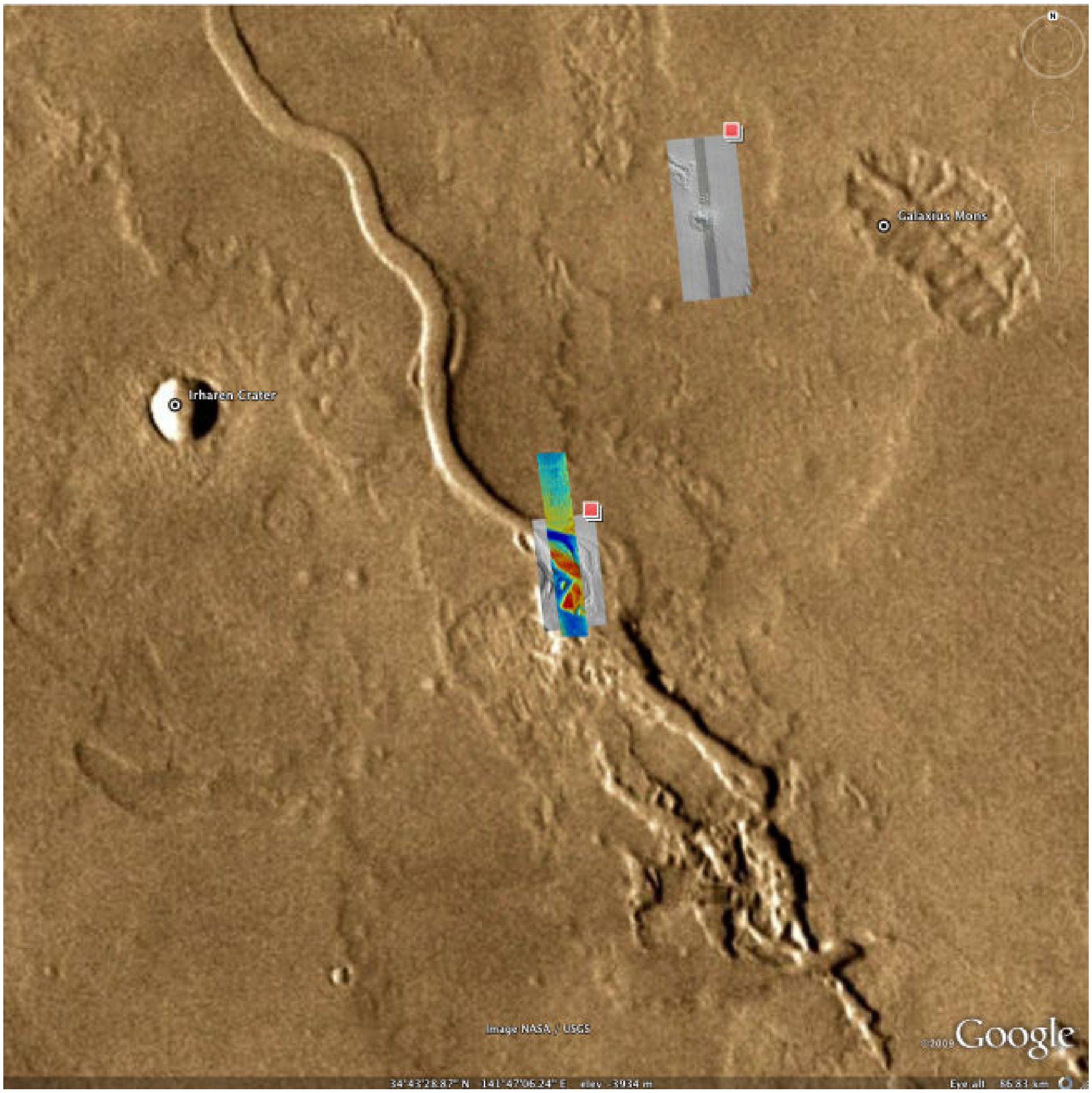


Figure 5.7: The colorized hillshade DEM as a KML overlay.

Part II

The Stereo Pipeline in Depth

Chapter 6

Stereo Correlation

In this chapter we will dive much deeper into understanding the core algorithms in the Stereo Pipeline. We start with an overview of the five stages of stereo reconstruction. Then we move into an in-depth discussion and exposition of the various correlation algorithms.

The goal of this chapter is to build an intuition for the stereo correlation process. This will help users to identify unusual results in their DEMs and hopefully eliminate them by tuning various parameters in the `stereo.default` file (appendix B). For scientists and engineers who are using DEMs produced with the Stereo Pipeline, this chapter may help to answer the question, "What is the Stereo Pipeline doing to the raw data to produce this DEM?"

A related question that is commonly asked is, "How accurate is a DEM produced by the Stereo Pipeline?" This chapter does not yet address matters of accuracy and error, however we have several efforts underway to quantify the accuracy of Stereo Pipeline-derived DEMs, and will be publishing more information about that shortly. Stay tuned.

The entire stereo correlation process, from raw input images to a point cloud or DEM, can be viewed as a multistage pipeline as depicted in Figure 6.1, and detailed in the following sections.

6.1 Pre-Processing

The first optional (but recommended) step in the process is least squares Bundle Adjustment, which is described in detail in Chapter 7.

Next, the left and right images are roughly aligned using one of the four methods: (1) a homography transform of the right image based on automated tie-point measurements, (2) Affine epipolar transform of both the left and right images (also based on tie-point measurements as earlier), the effect of which is equivalent to rotating the original cameras which took the pictures, (3) a 3D rotation that achieves epipolar rectification *(only implemented for Pinhole sessions for missions like MER or K10)* or (4) map-projection of both the left and right images using the ISIS `cam2map` command, or through `mapproject` for Digital Globe and GeoEye images (see section 4.2 for the latter). The first three options can be applied automatically by the Stereo Pipeline when the `alignment-method` variable in the `stereo.default` file is set to `affineepipolar`, `homography`, or `epipolar`, respectively.

The latter option, running `cam2map`, `cam2map4stereo.py`, or `mapproject` must be carried out by the user prior to invoking the `stereo` command. Map-projecting the images using ISIS eliminates any unusual distortion in the image due to the unusual camera acquisition modes (e.g. pitching "ROTO" maneuvers during image acquisition for MOC, or highly elliptical orbits and changing line exposure times for the High Resolution Stereo Camera, HRSC). It also eliminates some of the perspective differences in the image

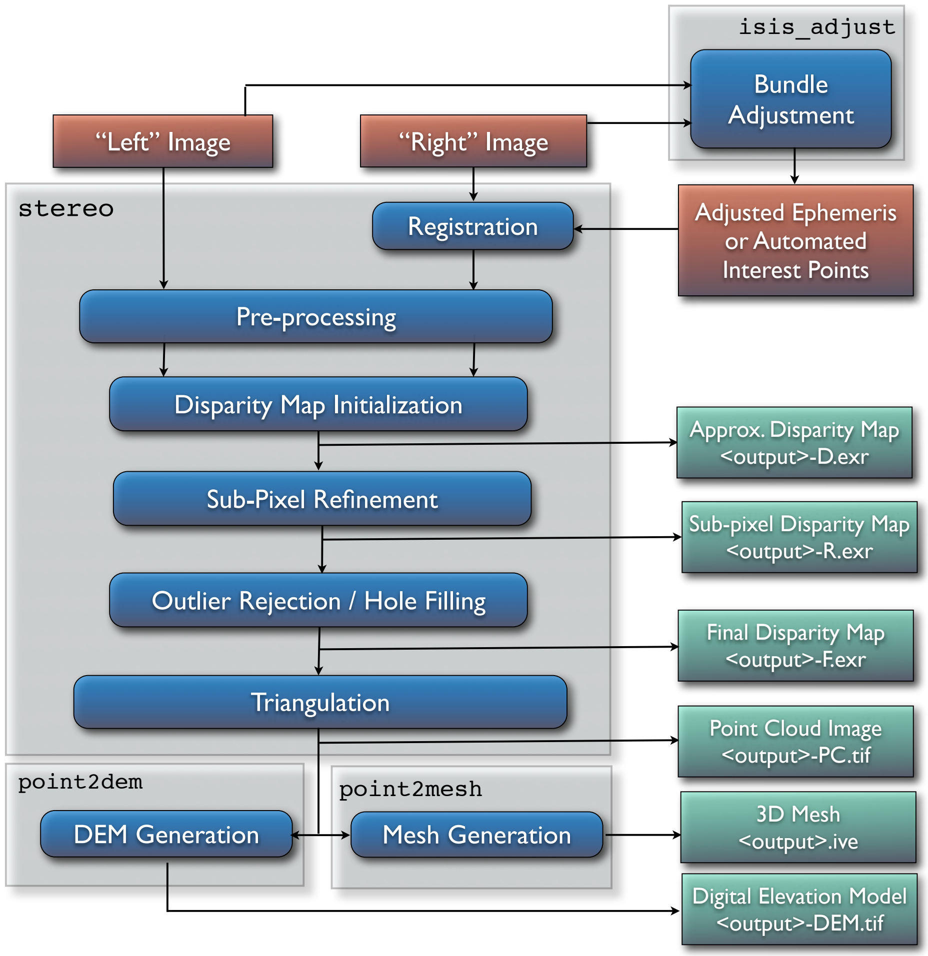


Figure 6.1: Flow of data through the Stereo Pipeline.

pair that are due to large terrain features by taking the existing low-resolution terrain model into account (e.g., the Mars Orbiter Laser Altimeter, MOLA; Lunar Orbiter Laser Altimeter, LOLA; National Elevation Dataset, NED; or Unified Lunar Coordinate Network, ULCN, 2005 models).

In essence, map-projecting the images results in a pair of very closely matched images that are as close to ideal as possible given existing information. This leaves only small perspective differences in the images, which are exactly the features that the stereo correlation process is designed to detect.

For this reason, we recommend map-projection for pre-alignment of most stereo pairs. Its only cost is longer triangulation times as more math must be applied to work back through the transforms applied to the images. In either case, the pre-alignment step is essential for performance because it ensures that the disparity search space is bounded to a known area. In both cases, the effects of pre-alignment are taken into account later in the process during triangulation, so you do not need to worry that pre-alignment will compromise the geometric integrity of your DEM.

In some cases the pre-processing step may also normalize the pixel values in the left and right images to bring them into the same dynamic range. Various options in the `stereo.default` file affect whether or how normalization is carried out, including `individually-normalize` and `force-use-entire-range`. Although the defaults work in most cases, the use of these normalization steps can vary from data set to data set, so we recommend you refer to the examples in Chapter 8 to see if these are necessary in your use case.

Finally, pre-processing can perform some filtering of the input images (as determined by prefilter-mode) to reduce noise and extract edges in the images. When active, these filters apply a kernel with a sigma of prefilter-kernel-width pixels that can improve results for noisy images (prefilter-mode must be chosen carefully in conjunction with cost-mode, see Appendix B). The pre-processing modes that extract image edges are useful for stereo pairs that do not have the same lighting conditions, contrast, and absolute brightness [24]. We recommend that you use the defaults for these parameters to start with, and then experiment only if your results are sub-optimal.

6.2 Disparity Map Initialization

Correlation is the process at the heart of the Stereo Pipeline. It is a collection of algorithms that compute correspondences between pixels in the left image and pixels in the right image. The map of these correspondences is called a *disparity map*. You can think of a disparity map as an image whose pixel locations correspond to the pixel (u, v) in the left image, and whose pixel values contain the horizontal and vertical offsets (d_u, d_v) to the matching pixel in the right image, which is $(u + d_u, v + d_v)$.

The correlation process attempts to find a match for every pixel in the left image. The only pixels skipped are those marked invalid in the mask images. For large images (e.g. from HiRISE, Lunar Reconnaissance Orbiter Camera, LROC, or WorldView), this is very expensive computationally, so the correlation process is split into two stages. The disparity map initialization step computes approximate correspondences using a pyramid-based search that is highly optimized for speed, but trades resolution for speed. The results of disparity map initialization are integer-valued disparity estimates. The sub-pixel refinement step takes these integer estimates as initial conditions for an iterative optimization and refines them using the algorithm discussed in the next section.

We employ several optimizations to accelerate disparity map initialization: (1) a box filter-like accumulator that reduces duplicate operations during correlation [28]; (2) a coarse-to-fine pyramid based approach where disparities are estimated using low-resolution images, and then successively refined at higher resolutions; and (3) partitioning of the disparity search space into rectangular sub-regions with similar values of disparity determined in the previous lower resolution level of the pyramid [28].

Naive correlation itself is carried out by moving a small, rectangular template window from the from left image over the specified search region of the right image, as in Figure 6.2. The "best" match is determined by applying a cost function that compares the two windows. The location at which the window evaluates to the lowest cost compared to all the other search locations is reported as the disparity value. The cost-mode variable allows you to choose one of three cost functions, though we recommend normalized cross correlation [19], since it is most robust to slight lighting and contrast variations between a pair of images. Try the others if you need more speed at the cost of quality.

Our implementation of pyramid correlation is a little unique in that it is actually split into two levels of pyramid searching. There is a *output_prefix*-D_sub.tif disparity image that is computed from the greatly reduced input images *-L_sub.tif and *output_prefix*-R_sub.tif. Those "sub" images have their size chosen so that their area is around 2.25 mega pixels, a size that is easily viewed on the screen unlike the raw source imagery. The low-resolution disparity image then defines the per thread search range of the higher resolution disparity, *output_prefix*-D.tif.

This solution is imperfect but comes from our model of multi-threaded processing. ASP processes individual tiles of the output disparity in parallel. The smaller the tiles, the easier it is to distribute evenly among the CPU cores. The size of the tile unfortunately limits the max number of pyramid levels we can process. We've struck a balance where every 1024 by 1024 pixel area is processed individually in a tile. This practice allows only 5 levels of pyramid processing. With the addition of the second tier of pyramid searching with *output_prefix*-D_sub.tif, we are allowed to process beyond that limitation.

Any large failure in the low-resolution disparity image will be detrimental to the performance of the higher

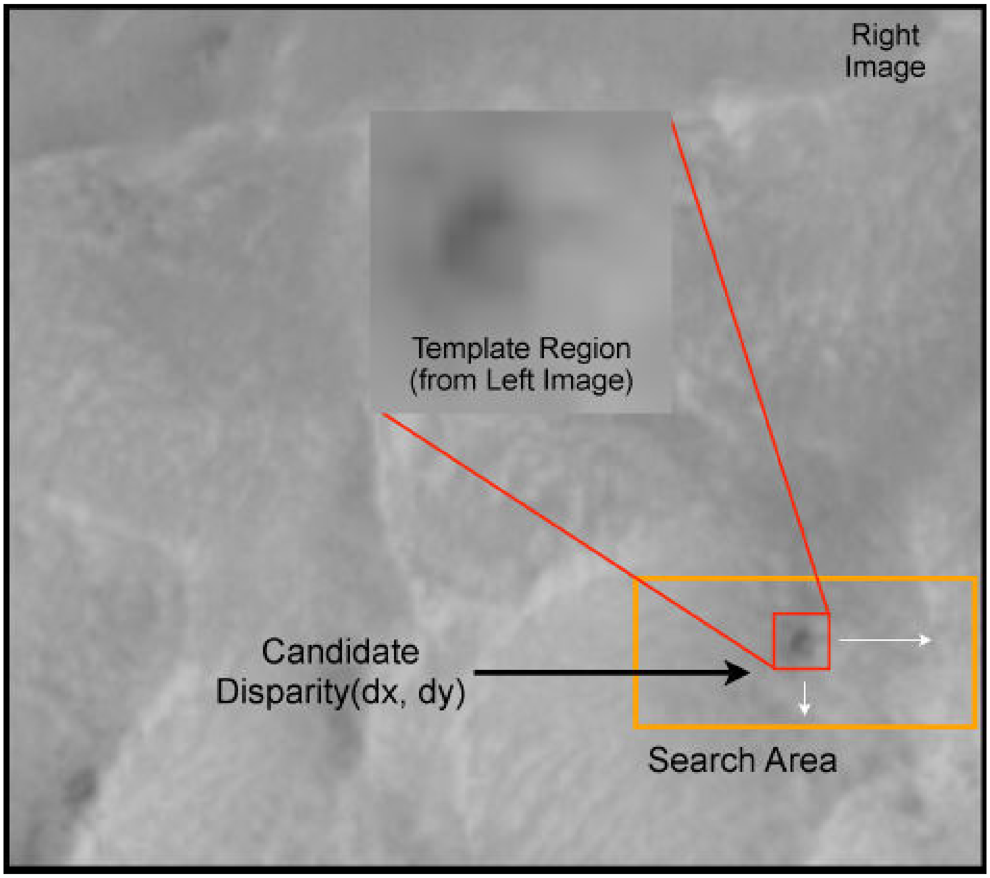


Figure 6.2: The correlation algorithm in disparity map initialization uses a sliding template window from the left image to find the best match in the right image. The size of the template window can be adjusted using the H_KERN and V_KERN parameters in the stereo.default file, and the search range can be adjusted using the {H,V}_CORR_{MIN/MAX} parameters.

resolution disparity. In the event that the low-resolution disparity is completely unhelpful, it can be skipped by adding `corr-seed-mode 0` in the stereo.default file. This should only be considered in cases where the texture in an image is completely lost when subsampled. An example would be satellite imagery of fresh snow in the Arctic. Alternatively, *output_prefix*-D_sub.tif can be computed at a sparse set of pixels at full resolution, as described in section 4.4.

An alternative to computing *output_prefix*-D.tif from sub-sampled images (`corr-seed-mode 1`) or skipping it altogether (`corr-seed-mode 0`), is to compute it from a lower-resolution DEM of the area (`corr-seed-mode 2`). In this situation, the low-resolution DEM needs to be specified together with its estimated error. See section B.2 for more detailed information as to how to specify these options. In our experiments, if the input DEM has a resolution of 1 km, a good value for the DEM error is about 10 m, or higher if the terrain is very variable.

6.2.1 Debugging Disparity Map Initialization

Never will all pixels be successfully matched during stereo matching. Though a good chunk of the image should be correctly processed. If you see large areas where matching failed, this could be due to a variety of reasons:

- In regions where the images do not overlap, there should be no valid matches in the disparity map.

- Match quality may be poor in regions of the images that have different lighting conditions, contrast, or specular properties of the surface.
- Areas that have image content with very little texture or extremely low contrast may have an insufficient signal to noise ratio, and will be rejected by the correlator.
- Areas that are highly distorted due to different image perspective, such as crater and canyon walls, may exhibit poor matching performance. This could also be due to failure of the preprocessing step in aligning the images. The correlator can not match images that are rotated differently from each other or have different scale/resolution.

Bad matches, often called "blunders" or "artifacts" are also common, and can happen for many of the same reasons listed above. The Stereo Pipeline does its best to automatically detect and eliminate these blunders, but the effectiveness of these outlier rejection strategies does vary depending on the quality of the input imagery.

When tuning up your `stereo.default` file, you will find that it is very helpful to look at the raw output of the disparity map initialization step. This can be done using the `disparitydebug` tool, which converts the *output_prefix*`-D.tif` file into a pair of normal images that contain the horizontal and vertical components of disparity. You can open these in a standard image viewing application and see immediately which pixels were matched successfully, and which were not. Stereo matching blunders are usually also obvious when inspecting these images. With a good intuition for the effects of various `stereo.default` parameters and a good intuition for reading the output of `disparitydebug`, it is possible to quickly identify and address most problems.

6.2.2 Local Homography

Correlation works by decomposing the left image into tiles, and for each pixel in each tile finding the best-matching pixel in the right image.

Depending on user's choices, by this stage either the left or the right image (or both) may already be transformed so that they are very similar, making the matching process more likely to succeed.

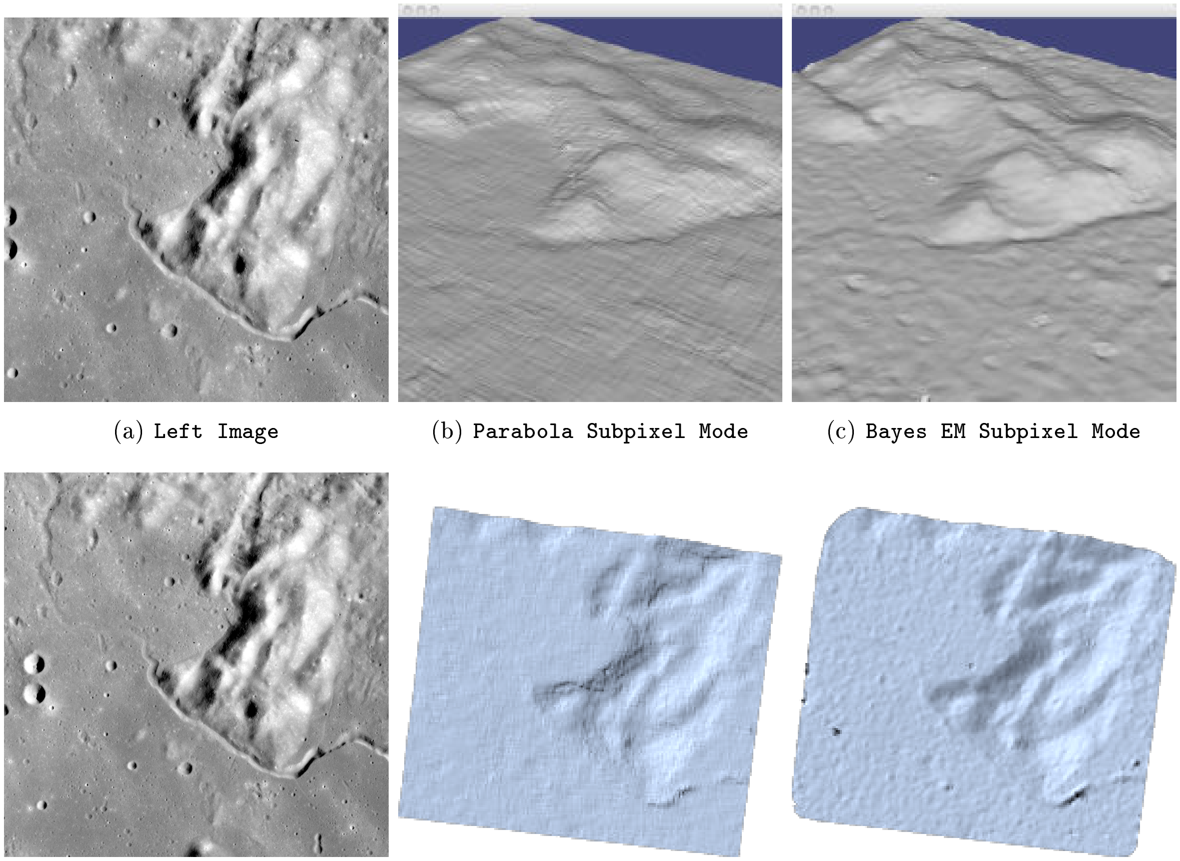
Whether that is the case or not, Stereo Pipeline can estimate, based on the low-resolution disparity *output_prefix*`-D_sub.tif`, a local homography transform for every left image tile, which, when applied to the right image, improves the similarity of the right image to the current left image tile. This option can be turned on with the flag `use-local-homography`.

This local homography transform comes in most useful when a global homography transform could not be applied (for example, if interest point matching failed). The input low-resolution disparity can be computed in several ways, as described earlier in the section.

6.3 Sub-pixel Refinement

Once disparity map initialization is complete, every pixel in the disparity map will either have an estimated disparity value, or it will be marked as invalid. All valid pixels are then adjusted in the sub-pixel refinement stage based on the `subpixel-mode` setting.

The first mode is parabola-fitting sub-pixel refinement (`subpixel-mode 1`). This technique fits a 2D parabola to points on the correlation cost surface in an 8-connected neighborhood around the cost value that was the "best" as measured during disparity map initialization. The parabola's minimum can then be computed analytically and taken as as the new sub-pixel disparity value.



(a) `Left Image` (b) `Parabola Subpixel Mode` (c) `Bayes EM Subpixel Mode`

(d) `Right Image` (e) `Parabola Hillshade` (f) `Bayes EM Hillshade`

Figure 6.3: Left: Input images. Center: results using the parabola draft subpixel mode (`subpixel-mode = 1`). Right: results using the Bayes EM high quality subpixel mode (`subpixel-mode = 2`).

This method is easy to implement and extremely fast to compute, but it exhibits a problem known as pixel-locking: the sub-pixel disparities tend toward their integer estimates and can create noticeable "stair steps" on surfaces that should be smooth [27, 29]. See for example Figure 6.3(b). Furthermore, the parabola subpixel mode is not capable of refining a disparity estimate by more than one pixel, so although it produces smooth disparity maps, these results are not much more accurate than the results that come out of the disparity map initialization in the first place. However, the speed of this method makes it very useful as a "draft" mode for quickly generating a DEM for visualization (i.e. non-scientific) purposes. It is also beneficial in the event that a user will simply downsample their DEM after generation in Stereo Pipeline.

For high quality results, we recommend `subpixel-mode 2`: the Bayes EM weighted affine adaptive window correlator. This advanced method produces extremely high quality stereo matches that exhibit a high degree of immunity to image noise. For example Apollo Metric Camera images are affected by two types of noise inherent to the scanning process: (1) the presence of film grain and (2) dust and lint particles present on the film or scanner. The former gives rise to noise in the DEM values that wash out real features, and the latter causes incorrect matches or hard to detect blemishes in the DEM. Attenuating the effect of these scanning artifacts while simultaneously refining the integer disparity map to sub-pixel accuracy has become a critical goal of our system, and is necessary for processing real-world data sets such as the Apollo Metric Camera data.

The Bayes EM subpixel correlator also features a deformable template window from the left image that can be rotated, scaled, and translated as it zeros in on the correct match in the right image. This adaptive window is essential for computing accurate matches on crater or canyon walls, and on other areas with significant perspective distortion due to foreshortening.

This affine-adaptive behavior is based on the Lucas-Kanade template tracking algorithm, a classic algorithm in the field of computer vision [3]. We have extended this technique; developing a Bayesian model that treats the Lucas-Kanade parameters as random variables in an Expectation Maximization (EM) framework. This statistical model also includes a Gaussian mixture component to model image noise that is the basis for the robustness of our algorithm. We will not go into depth on our approach here, but we encourage interested readers to read our papers on the topic [23, 5].

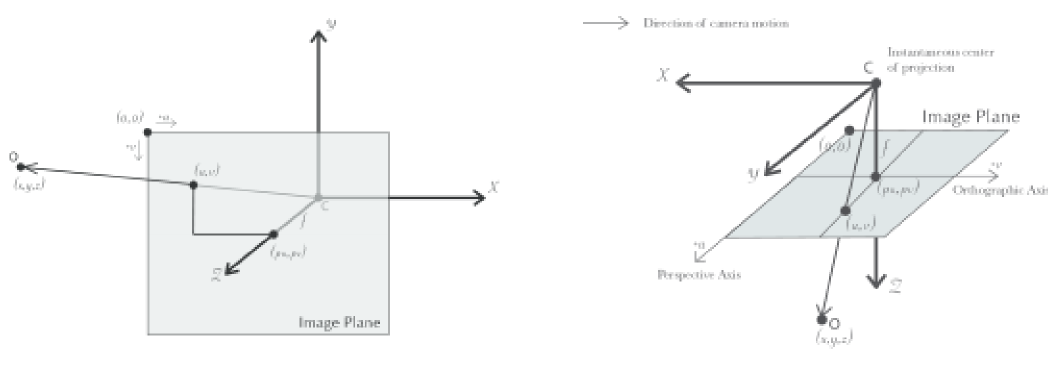
However we do note that, like the computations in the disparity map initialization stage, we adopt a multi-scale approach for sub-pixel refinement. At each level of the pyramid, the algorithm is initialized with the disparity determined in the previous lower resolution level of the pyramid, thereby allowing the subpixel algorithm to shift the results of the disparity initialization stage by many pixels if a better match can be found using the affine, noise-adapted window. Hence, this sub-pixel algorithm is able to significantly improve upon the results to yield a high quality, high resolution result.

Another option when run time is important is `subpixel-mode 3`: the simple affine correlator. This is essentially the Bayes EM mode with the noise correction features removed in order to decrease the required run time. In data sets with little noise this mode can yield results similar to Bayes EM mode in approximately one fifth the time.

6.4 Triangulation

When running an ISIS session, the Stereo Pipeline uses geometric camera models available in ISIS [2]. These highly accurate models are customized for each instrument that ISIS supports. Each ISIS "cube" file contains all of the information that is required by the Stereo Pipeline to find and use the appropriate camera model for that observation.

Other sessions such as DG (*Digital Globe*) or Pinhole, require that their camera model be provided as additional arguments to the `stereo` command. Those camera models come in the form of an XML document for DG and as `*.pinhole, *.tsai, *.cahv, *.cahvor` for Pinhole sessions. Those files must be the third and forth arguments or immediately follow after the 2 input images for `stereo`.



(a) `Framing Camera Model` (b) `Pushbroom Camera Model`

Figure 6.4: Most remote sensing cameras fall into two generic categories based on their basic geometry. Framing cameras (left) capture an instantaneous two-dimensional image. Linescan cameras (right) capture images one scan line at a time, building up an image over the course of several seconds as the satellite moves through the sky.

ISIS camera models account for all aspects of camera geometry, including both intrinsic (i.e. focal length,

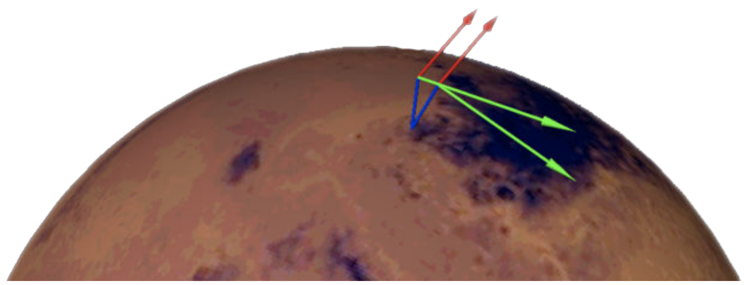


Figure 6.5: Once a disparity map has been generated and refined, it can be used in combination with the geometric camera models to compute the locations of 3D points on the surface of Mars. This figure shows the position (at the origins of the red, green, and blue vectors) and orientation of the Mars Global Surveyor at two points in time where it captured images in a stereo pair.

pixel size, and lens distortion) and extrinsic (e.g. camera position and orientation) camera parameters. Taken together, these parameters are sufficient to "forward project" a 3D point in the world onto the image plane of the sensor. It is also possible to "back project" from the camera's center of projection through a pixel corresponding to the original 3D point.

Notice, however, that forward and back projection are not symmetric operations. One camera is sufficient to "image" a 3D point onto a pixel located on the image plane, but the reverse is not true. Given only a single camera and a pixel location $x = (u, v)$, that is the image of an unknown 3D point $P = (x, y, z)$, it is only possible to determine that P lies somewhere along a ray that emanates from the camera's center of projection through the pixel location x on the image plane (see Figure 6.4).

Alas, once images are captured, the route from image pixel back to 3D points in the real world is through back projection, so we must bring more information to bear on the problem of uniquely reconstructing our 3D point. In order to determine P using back projection, we need *two* cameras that both contain pixel locations x_1 and x_2 where P was imaged. Now, we have two rays that converge on a point in 3D space (see Figure 6.5). The location where they meet must be the original location of P.

In practice, the two rays rarely intersect perfectly because any slight error in the camera position or pointing information will effect the rays' positions as well. Instead, we take the *closest point of intersection* of the two rays as the location of point P.

Additionally, the actual distance between the rays at this point is an interesting and important error metric that measures how self-consistent our two camera models are for this point. You will learn in the next chapter that this information, when computed and averaged over all reconstructed 3D points, can be a valuable statistic for determining whether to carry out bundle adjustment. Distance between the two rays at their closest intersection is recorded in the fourth channel of the point cloud file, ***output-prefix***`-PC.tif`. This information can be brought to the same perspective as the output DEM by using the *--error* argument on the `point2dem` command.

This error in the triangulation, the distance between two rays, *is not the true accuracy of the DEM.* It is only another indirect measure of quality. A DEM with high triangulation error is always bad and should have its images bundle-adjusted. A DEM with low triangulation error is at least self consistent but could still be bad. A map of the triangulation error should only be interpreted as a relative measurement. Where small areas are found with high triangulation error came from correlation mistakes and large areas of error came from camera model inadequacies.

Chapter 7

Bundle Adjustment

7.1 Overview

Satellite position and orientation errors have a direct effect on the accuracy of digital elevation models produced by the Stereo Pipeline. If they are not corrected, these uncertainties will result in systematic errors in the overall position and slope of the DEM. Severe distortions can occur as well, resulting in twisted or “taco shaped” DEMs, though in most cases these effects are quite subtle and hard to detect. In the worst case, such as with old mission data like Voyager or Apollo, these gross camera misalignments can inhibit Stereo Pipeline’s internal interest point matcher and block auto search range detection.

Errors in camera position and orientation can be corrected using a process called *bundle adjustment*. Bundle adjustment is the process of simultaneously adjusting the properties of many cameras and the 3D locations of the objects they see in order to minimize the error between the estimated, back-projected pixel locations of the 3D objects and their actual measured locations in the captured images.

This complex process can be boiled down to this simple idea: bundle adjustment ensures that the observations in multiple images of a single ground feature are self-consistent. If they are not consistent, then the position and orientation of the cameras as well as the 3D position of the feature must be adjusted until they are. This optimization is carried out along with thousands (or more) of similar constraints involving many different features observed in other images. Bundle adjustment is very powerful and versatile: it can operate on just two overlapping images, or on thousands. It is also a dangerous tool. Careful consideration

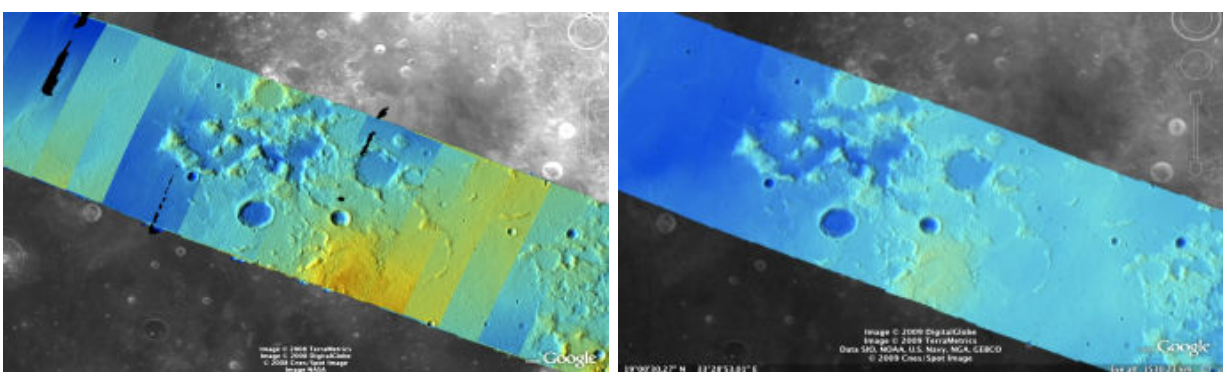


Figure 7.1: Bundle adjustment is illustrated here using a color-mapped, hill-shaded DEM mosaic from Apollo 15, Orbit 33, imagery. (a) Prior to bundle adjustment, large discontinuities can exist between overlapping DEMs made from different images. (b) After bundle adjustment, DEM alignment errors are minimized and no longer visible.

is required to insure and verify that the solution does represent reality.

Bundle adjustment can also take advantage of ground control points (GCPs), which are 3D locations of features that are known a priori (often by measuring them by hand in another existing DEM). GCPs can improve the internal consistency of your DEM or align your DEM to an existing data product. Finally, even though bundle adjustment calculates the locations of the 3D objects it views, only the final properties of the cameras are recorded for use by the Ames Stereo Pipeline. Those properties can be loaded into the `stereo` program which uses its own method for triangulating 3D feature locations.

When using the Stereo Pipeline, bundle adjustment is an optional step between the capture of images and the creation of DEMs. The bundle adjustment process described below should be completed prior to running the `stereo` command.

Although bundle adjustment is not a required step for generating DEMs, it is *highly recommended* for users who plan to create DEMs for scientific analysis and publication. Incorporating bundle adjustment into the stereo work flow not only results in DEMs that are more internally consistent, it is also the correct way to co-register your DEMs with other existing data sets and geodetic control networks.

At the moment however, Bundle Adjustment does not automatically work against outside DEMs from sources such as laser altimeters. Hand-picked GCPs are the only way for ASP to register to those types of sources.

7.2 Bundle adjustment using ASP

Recently, Stereo Pipeline started providing its own bundle adjustment tool, named `bundle_adjust`. Its usage is described in section A.3.

Here is an example of using this tool on a couple of Apollo 15 images, and its effect on decreasing the stereo triangulation error.

Running `stereo` without using bundle-adjusted camera models.

```
stereo AS15-M-1134.cub AS15-M-1135.cub run_noadjust/run
```

Performing bundle adjustment.

```
bundle_adjust AS15-M-1134.cub AS15-M-1135.cub -o run_ba/run
```

Running stereo while using the bundle-adjusted camera models.

```
stereo AS15-M-1134.cub AS15-M-1135.cub run_adjust/run \
  --bundle-adjust-prefix run_ba/run
```

A comparison of the two ways of doing stereo is shown in figure 7.2.

7.3 Bundle adjustment using ISIS

In what follows we describe how to do bundle adjustment using ISIS's toolchain. It also serves to describe bundle adjustment in more detail, which is applicable to other bundle adjustment tools as well, including Stereo Pipeline's own tool.

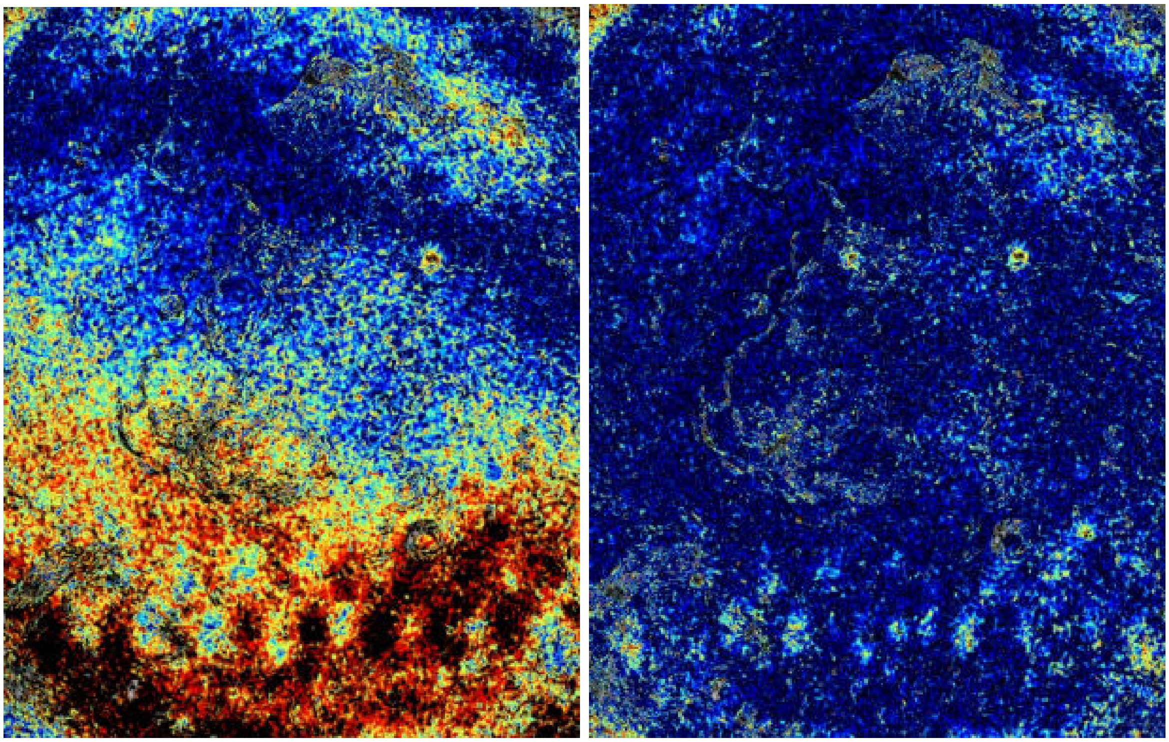


Figure 7.2: Illustration of the triangulation error map for a pair of images before (left) and after (right) using Stereo Pipeline's `bundle_adjust`. Red and black colors suggest higher error.

In bundle adjustment, the position and orientation of each camera station are determined jointly with the 3D position of a set of image tie-points points chosen in the overlapping regions between images. Tie points, as suggested by the name, tie multiple camera images together. Their physical manifestation would be a rock or small crater than can be observed across more than one image.

Tie-points are automatically extracted using ISIS's `autoseed` and `pointreg` (alternatively one could use a number of outside methods such as the famous SURF[4]). Creating a collection of tie points, called a *control network*, is a three step process. First, a general geographic layout of the points must be decided upon. This is traditionally just a grid layout that has some spacing that allows for about 20-30 measurements to be made per image. This shows up in slightly different projected locations in each image due to their slight misalignments. The second step is to have an automatic registration algorithm try to find the same feature in all images using the prior grid as a starting location. The third step is to manually verify all measurements visually, checking to insure that each measurement is looking at the same feature.

Bundle Adjustment in ISIS is performed with the `jigsaw` executable. It generally follows the method described in [30] and determines the best camera parameters that minimize the projection error given by $\epsilon = \sum_k \sum_j (I_k - I(C_j, X_k))^2$ where I_k are the tie points on the image plane, C_j are the camera parameters, and X_k are the 3D positions associated with features I_k. $I(C_j, X_k)$ is an image formation model (i.e. forward projection) for a given camera and 3D point. To recap, it projects the 3D point, X_k, into the camera with parameters C_j. This produces a predicted image location for the 3D point that is compared against the observed location, I_k. It then reduces this error with the Levenberg-Marquardt algorithm (LMA). Speed is improved by using sparse methods as described in Hartley and Zisserman [14], Konolige [15], and Chen et al. [7].

Even though the arithmetic for bundle adjustment sounds clever, there are faults with the base implementation. Imagine a case where all cameras and 3D points were collapsed into a single point. If you evaluate the above cost function, you'll find that the error is indeed zero. This is not the correct solution if the images were taken from orbit. Another example is if a translation was applied equally to all 3D points and camera locations. This again would not affect the cost function. This fault comes from bundle adjustment's inability to control the scale and translation of the solution. It will correct the geometric shape of the problem, yet it cannot guarantee that the solution will have correct scale and translation.

ISIS attempts to fix this problem by adding two additional cost functions to bundle adjustment. First of which is $\epsilon = \sum_j (C_j^{initial} - C_j)^2$. This constrains camera parameters to stay relatively close to their initial values. Second, a small handful of 3D ground control points can be chosen by hand and added to the error metric as $\epsilon = \sum_k (X_k^{gcp} - X_k)^2$ to constrain these points to known locations in the planetary coordinate frame. A physical example of a ground control point could be the location of a lander that has a well known location. GCPs could also be hand-picked points against a highly regarded and prior existing map such as the THEMIS Global Mosaic or the LRO-WAC Global Mosaic.

Like other iterative optimization methods, there are several conditions that will cause bundle adjustment to terminate. When updates to parameters become insignificantly small or when the error, ϵ, becomes insignificantly small, then the algorithm has converged and the result is most likely as good as it will get. However, the algorithm will also terminate when the number of iterations becomes too large in which case bundle adjustment may or may not have finished refining the parameters of the cameras.

7.3.1 Tutorial: Processing Mars Orbital Camera Imagery

This tutorial for ISIS's bundle adjustment tools is taken from [21] and [22]. These tools are not a product of NASA nor the authors of Stereo Pipeline. They were created by USGS and their documentation is available

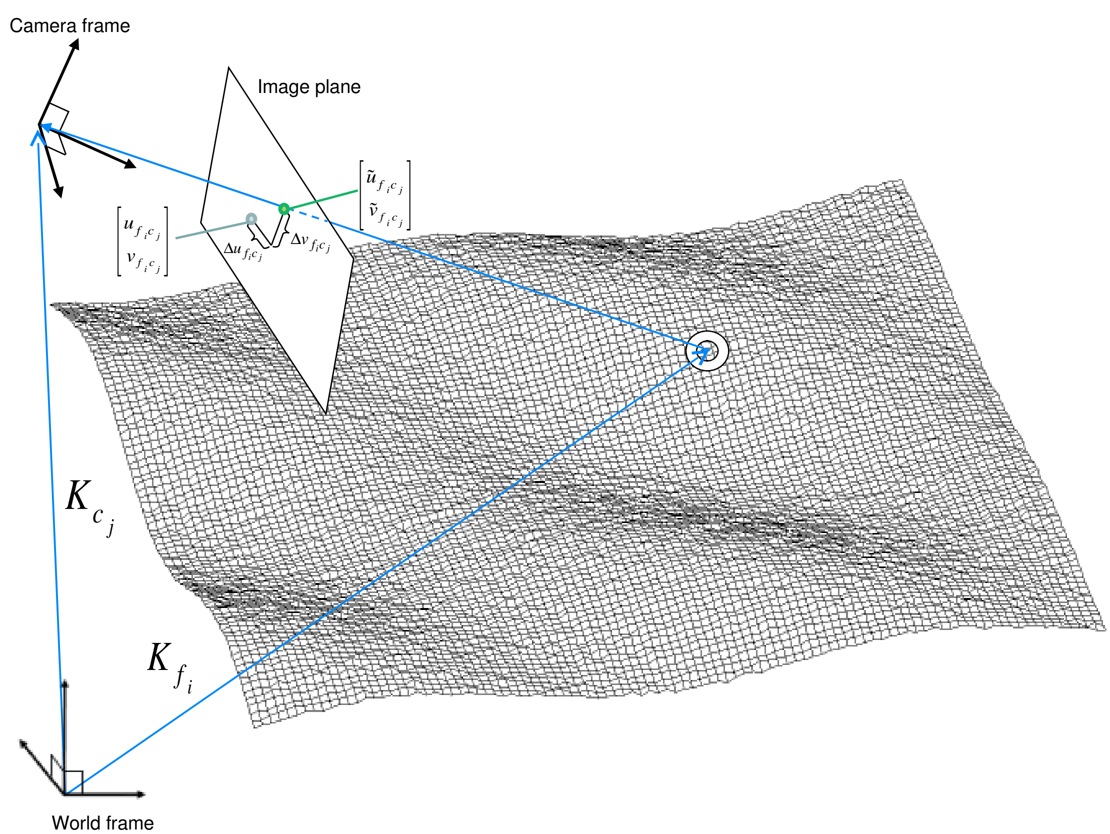


Figure 7.3: A feature observation in bundle adjustment, from Moore et al. [20]

at [6].

What follows is an example of bundle adjustment using two MOC images of Hrad Vallis. We use images E02/01461 and M01/00115, the same as used in Chapter 3. These images are available from NASA's PDS (the ISIS `mocproc` program will operate on either the IMQ or IMG format files, we use the `.imq` below in the example). For reference, the following ISIS commands are how to convert the MOC images to ISIS cubes.

```
ISIS 3> mocproc from=e0201461.imq to=e0201461.cub mapping=no
ISIS 3> mocproc from=m0100115.imq to=m0100115.cub mapping=no
```

Note that the resulting images are not map-projected. Bundle adjustment requires the ability to project arbitrary 3D points into the camera frame. The process of map-projecting an image dissociates the camera model from the image. Map-projecting can be perceived as the generation of a new infinitely large camera sensor that may be parallel to the surface, a conic shape, or something more complex. That makes it extremely hard to project a random point into the camera's original model. The math would follow the transformation from projection into the camera frame, then projected back down to surface that ISIS uses, then finally up into the infinitely large sensor. `Jigsaw` does not support this and thus does not operate on map-projected imagery.

Before we can dive into creating our tie-point measurements we must finish prepping these images. The following commands will add a vector layer to the cube file that describes its outline on the globe. It will also create a data file that describes the overlapping sections between files.

```
ISIS 3> footprintinit from=e0201461.cub
ISIS 3> footprintinit from=m0100115.cub
ISIS 3> echo *cub |  xargs -n1 echo > cube.lis
ISIS 3> findimageoverlaps from=cube.lis overlaplist=overlap.lis
```

At this point, we are ready to start generating our measurements. This is a three step process that requires defining a geographic pattern for the layout of the points on the groups, an automatic registration pass, and finally a manual clean up of all measurements. Creating the ground pattern of measurements is performed with `autoseed`. It requires a settings file that defines the spacing in meters between measurements. For this example, write the following text into a *autoseed.def* file.

```
Group = PolygonSeederAlgorithm
      Name = Grid
      MinimumThickness = 0.01
      MinimumArea = 1
      XSpacing = 1000
      YSpacing = 2000
End_Group
```

The minimum thickness defines the minimum ratio between the sides of the region that can have points applied to it. A choice of 1 would define a square and anything less defines thinner and thinner rectangles. The minimum area argument defines the minimum square meters that must be in an overlap region. The last two are the spacing in meters between control points. Those values were specifically chosen for this pair so that about 30 measurements would be produced from `autoseed`. Having more control points just makes for more work later on in this process. Run `autoseed` with the following instruction.

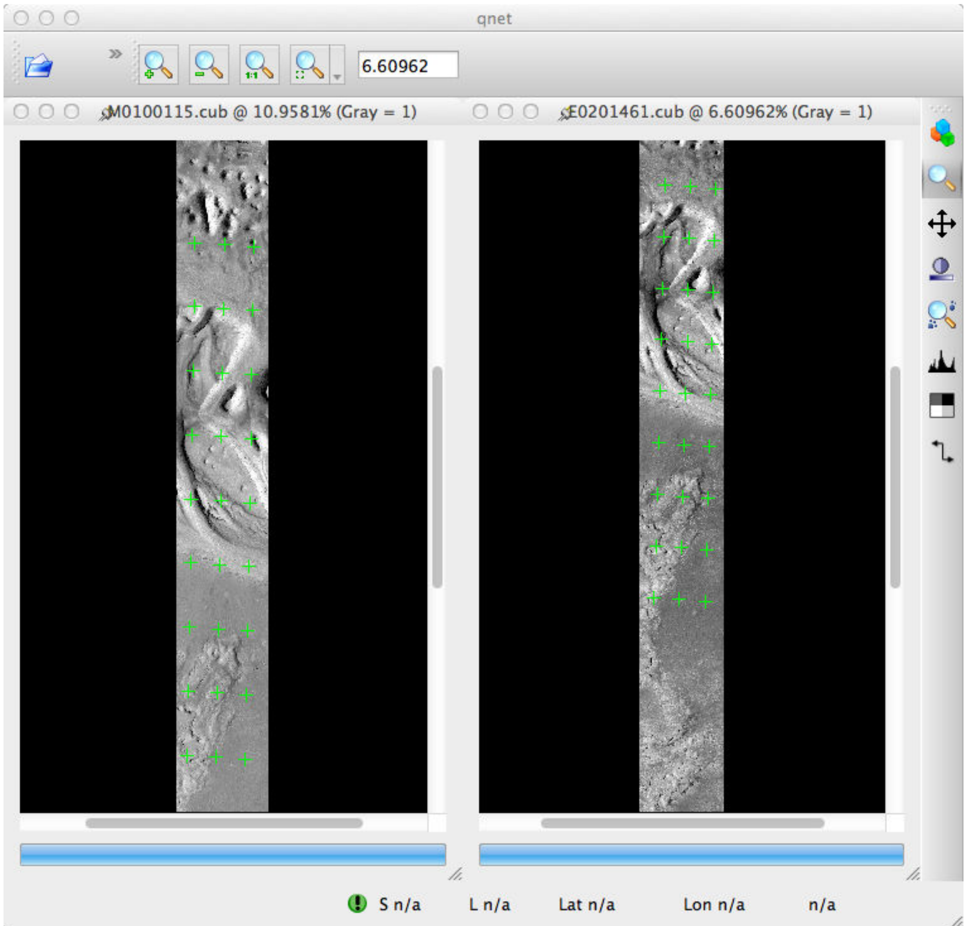


Figure 7.4: A visualization of the features laid out by `autoseed` in `qnet`. Note that the marks do not cover the same features between images. This is due to the poor initial spice data for MOC imagery.

```
ISIS 3> autoseed fromlist=cube.lis overlaplist=overlap.lis    \
          onet=control.net deffile=autoseed.def networkid=moc \
          pointid=???? description=hrad_vallis
```

The next step is to perform auto registration of these features between the two images using `pointreg`. This program also requires a settings file that describes how to do the automatic search. Copy the text box below into a *autoRegTemplate.def* file.

```
Object = AutoRegistration
 Group = Algorithm
   Name          = MaximumCorrelation
   Tolerance     = 0.7
 EndGroup

 Group = PatternChip
   Samples = 21
   Lines   = 21
   MinimumZScore = 1.5
   ValidPercent = 80
 EndGroup
```

```
  Group = SearchChip
    Samples = 75
    Lines   = 1000
  EndGroup
EndObject
```

The search chip defines the search range for which `pointreg` will look for matching imagery. The pattern chip is simply the kernel size of the matching template. The search range is specific for this image pair. The control network result after `autoseed` had a large vertical offset in the ball park of 500 px. The large misalignment dictated the need for the large search in the lines direction. Use `qnet` to get an idea for what the pixel shifts look like in your stereo pair to help you decide on a search range. In this example, only one measurement failed to match automatically. Here are the arguments to use in this example of `pointreg`.

```
ISIS 3> pointreg fromlist=cube.lis cnet=control.net              \
           onet=control_pointreg.net deffile=autoRegTemplate.def
```

The third step is to manually edit the control and verify the measurements in `qnet`. Type `qnet` in the terminal and then open *cube.lis* and lastly *control_pointreg.net*. From the Control Network Navigator window, click on the first point listed as *0001*. That opens a third window called the Qnet Tool. That window will allow you to play a flip animation that shows alignment of the feature between the two images. Correcting a measurement is performed by left clicking in the right image, then clicking *Save Measure*, and finally finishing by clicking *Save Point*.

In this tutorial, measurement *0025* ended up being incorrect. Your number may vary if you used different settings than the above or if MOC spice data has improved since this writing. When finished, go back to the main Qnet window. Save the final control network as *control_qnet.net* by clicking on *File*, and then *Save As*.

Once the control network is finished, it is finally time to start bundle adjustment. Here's what the call to `jigsaw` looks like:

```
ISIS 3> jigsaw fromlist=cube.lis update=yes twist=no radius=yes \
           cnet=control_qnet.net onet=control_ba.net
```

The update option defines that we would like to update the camera pointing, if our bundle adjustment converges. The *twist=no* says to not solve for the camera rotation about the camera bore. That property is usually very well known as it is critical for integrating an image with a line-scan camera. The *radius=yes* means that the radius of the 3D features can be solved for. Using no will force the points to use height values from another source, usually LOLA or MOLA.

The above command will spew out a bunch of diagnostic information from every iteration of the optimization algorithm. The most important feature to look at is the *sigma0* value. It represents the mean of pixel errors in the control network. In our run, the initial error was 1065 px and the final solution had an error of 1.1 px.

Producing a DEM using the newly created camera corrections is the same as covered in the Tutorial on page 15. When using `jigsaw`, it modifies a copy of the spice data that is stored internally to the cube file. Thus when we want to create a DEM using the correct camera geometry, no extra information needs to be given to `stereo` since it is already contained in the file. In the event a mistake has been made, `spiceinit` will overwrite the spice data inside a cube file and provide the original uncorrected camera pointing.

```
ISIS 3> stereo E0201461.cub M0100115.cub bundled/bundled
```

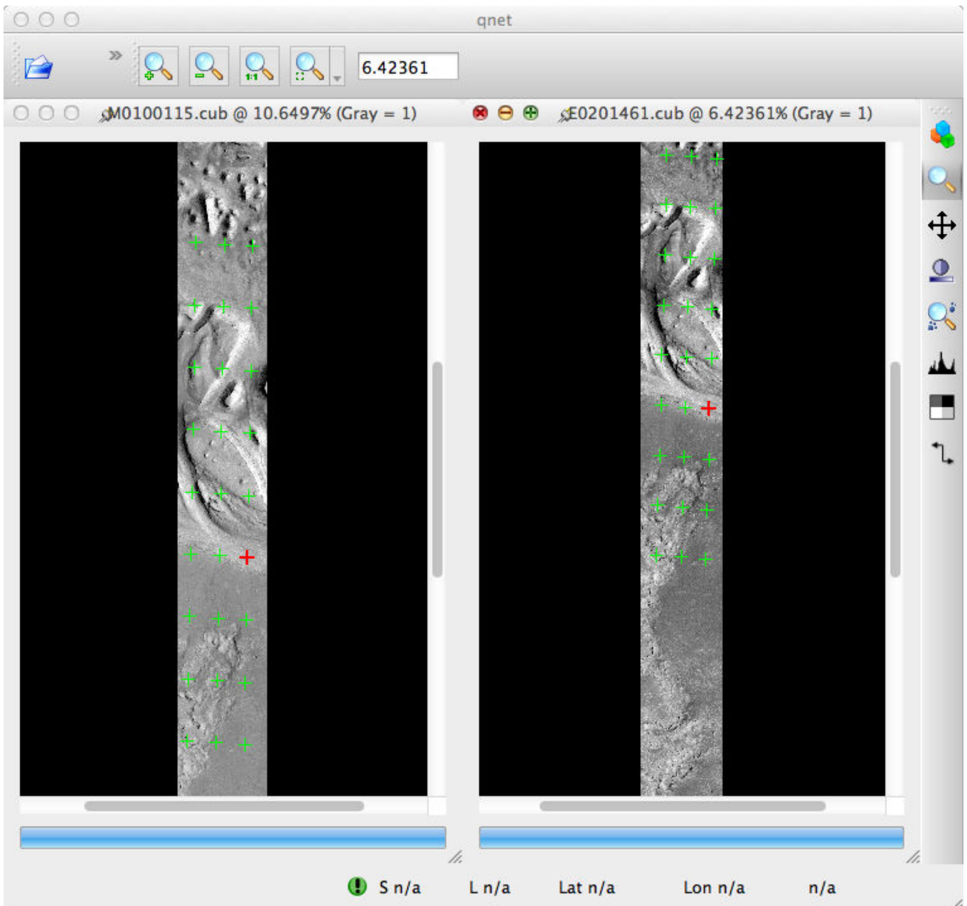


Figure 7.5: A visualization of the features after manual editing in `qnet`. Note that the marks now appear in the same location between images.

Chapter 8

Data Processing Examples

This chapter showcases a variety of results that are possible when processing different data sets with the Stereo Pipeline. It is also a shortened guide that shows the commands used to process specific mission data. There is no definitive method yet for making elevation models as each stereo pair is unique. We hope that the following sections serve as a cookbook for strategies that will get you started in processing your own data. We recommend that you second check your results against another source.

8.1 Guidelines for Selecting Stereo Pairs

When choosing image pairs to process, images that are taken with similar viewing angles, lighting conditions, and significant surface coverage overlap are best suited for creating terrain models. Depending on the characteristics of the mission data set and the individual images, the degree of acceptable variation will differ. Significant differences between image characteristics increases the likelihood of stereo matching error and artifacts, and these errors will propagate through to the resulting data products.

Although images do not need to be map-projected before running the `stereo` program, we recommend that you do run `cam2map` (or `cam2map4stereo.py`) beforehand, especially for image pairs that contain large topographic variation (and therefore large disparity differences across the scene, e.g., Valles Marineris). Map-projection is especially necessary when processing HiRISE images. This removes the large disparity differences between HiRISE images and leaves only the small detail for the Stereo Pipeline to compute. Remember that ISIS can work backwards through a map-projection when applying the camera model, so the geometric integrity of your images will not be sacrificed if you map-project first.

Excessively noisy images will not correlate well, so images should be photometrically calibrated in whatever fashion suits your purposes. If there are photometric problems with the images, those photometric defects can be misinterpreted as topography.

Remember, in order for `stereo` to process stereo pairs in ISIS cube format, the images must have had SPICE data associated by running ISIS's `spiceinit` program run on them first.

8.1.1 Combating Long Run Times

The factor that predominantly determines running time in the Stereo Pipeline is the size of the search space considered by the correlation algorithm. This is set in the `stereo.default` file using the `corr-search` parameter. If you comment that parameter out (either by putting a '`#`' at the beginning of their line or deleting them from your `stereo.default` file), the Stereo Pipeline will try to automatically determine the search range for you, but this does not always work perfectly. A spurious bad match can lead the pipeline to select a search range that is far too large, and performance will suffer as a result. If you know (or can estimate) the range of horizontal and vertical offsets you expect to see between the two images, then you may want to try setting the search range yourself in your `stereo.default` using the aforementioned parameters.

More generally, here are several strategies that tend to keep the search range small and run-times low:

1. You can instruct ASP to work only on a subregion of the left input image (section A.1). Run times will be much lower (minutes instead of days), and you can quickly tune parameters in the `stereo.default` file before scaling up to the full image.

2. You can use the `parallel_stereo` tool to distribute the computations over multiple machines (section A.2).

3. A solution specific to ISIS imagery is to crop your stereo pair (using the ISIS `crop` command) to a small region of interest within a large stereo pair.

4. The image pair can be subsampled. For ISIS imagery, the ISIS `reduce` command can be used, while for Digital Globe data one can invoke the `dg_mosaic` tool (section A.8). With subsampling, you are trading resolution for speed, so this probably only makes sense for debugging or "previewing" 3D terrain. That said, subsampling will tend to increase the signal to noise ratio, so it may also be helpful for obtaining 3D terrain out of noisy, low quality images.

   These options of cropping or reducing the resolution of the source imagery are only easily achieved with ISIS or Digital Globe data. For Pinhole or RPC sessions, users may reduce the image size using for example GDAL, but then the camera models will need to be adjusted manually. This is a unique problem for each camera model and thus will not be discussed here.

5. You can map-project the images. For Digital Globe images one can use `mapproject` (section 4.2), while for ISIS data the ISIS `cam2map` command or the `cam2map4stereo.py` program provided with the Stereo Pipeline can be applied. If you project both images into the same map-projection and same pixel scale, then they will be aligned modulo uncertainty in spacecraft telemetry (typically tens or hundreds of meters of error when the image is projected onto the ground). By default `cam2map` will also project the image onto the local elevation model (MOLA or LOLA), which removes the stereo disparity in the images that is due to coarse topography. The resulting image pair has only small position offsets and fine 3D detail left to discover, so the search range can be kept very small and run times can be improved. The Stereo Pipeline will keep track of how these map-projections affect the camera model, and take them into account when building up the 3D mesh via triangulation. If you use `cam2map`, be sure that your `stereo.default`'s `alignment-method` is set to `none`. Note also that the `--lat` and `--lon` arguments to `cam2map4stereo.py` can be used to crop your stereo images, and the `--resolution` argument can be used to subsample them.

If you are working with very large images, we highly recommend cropping or subsampling and working with smaller sized images while you fine-tune the parameters in the `stereo.default` file, and once you get satisfactory results to apply those parameters to the full images.

8.2 Mars Reconnaissance Orbiter HiRISE

HiRISE is one of the most challenging cameras to use when making 3D models because HiRISE exposures can be several gigabytes each. Working with this data requires patience as it will take time.

One important fact to know about HiRISE is that it is composed of multiple linear CCDs that are arranged side by side with some vertical offsets. These offsets mean that the CCDs will view some of the same terrain but at a slightly different time and a slightly different angle. Mosaicking the CCDs together to a single image is not a simple process and involves living with some imperfections.

One cannot simply use the HiRISE RDR products, as they do not have the required geometric stability. Instead, the HiRISE EDR products must be assembled using ISIS `noproj`. The USGS distributes a script in use by the HiRISE team that works forward from the team-produced 'balance' cubes, which provides a de-jittered, noproj'ed mosaic of a single observation, which is perfectly suitable for use by the Stereo Pipeline (this script was originally engineered to provide input for SOCET SET). However, the 'balance' cubes are not available to the general public, and so we include a program (`hiedr2mosaic.py`, written in Python) that will take PDS available HiRISE EDR products and walk through the processing steps required to provide good input images for `stereo`.

The program takes all the red CCDs and projects them using the ISIS `noproj` command into the perspective of the RED5 CCD. From there, `hijitreg` is performed to work out the relative offsets between CCDs. Finally the CCDs are mosaicked together using the average offset listed from `hijitreg` using the `handmos` command. Below is an outline of the processing.

```
hi2isis          # Import HiRISE IMG to Isis
hical            # Calibrate
histitch         # Assemble whole-CCD images from the channels
spiceinit
spicefit         # For good measure
noproj           # Project all images into perspective of RED5
hijitreg         # Work out alignment between CCDs
handmos          # Mosaic to single file
```

To use our script, first go to the directory where you have downloaded the HiRISE's RED EDR `IMG` files. You can run the `hiedr2mosaic.py` program without any arguments to view a short help statement, with the `-h` option to view a longer help statement, or just run the program on the EDR files like so:

```
hiedr2mosaic.py *.IMG
```

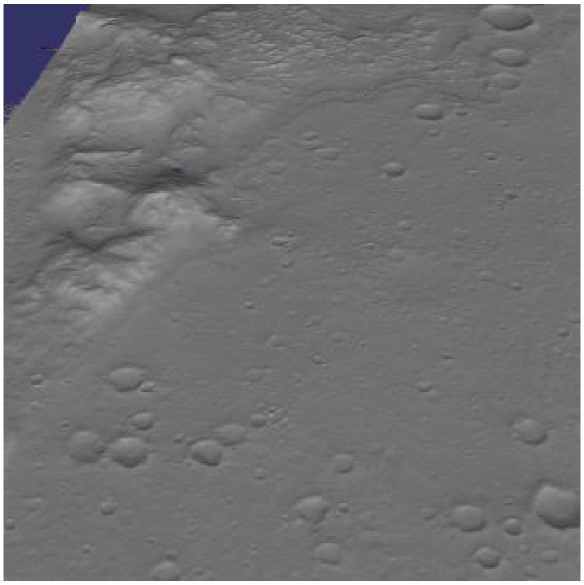
If you have more than one observation's worth of EDRs in that directory, then limit the program to just one observation's EDRs at a time, e.g. `hiedr2mosaic.py PSP_001513_1655*IMG`. If you run into problems, try using the `-k` option to retain all of the intermediary image files to help track down the issue. The `hiedr2mosaic.py` program will create a single mosaic file with the extension `.mos_hijitreged.norm.cub`. Be warned that the operations carried out by `hiedr2mosaic.py` can take many hours to complete on the very large HiRISE images.

8.2.1 Columbia Hills

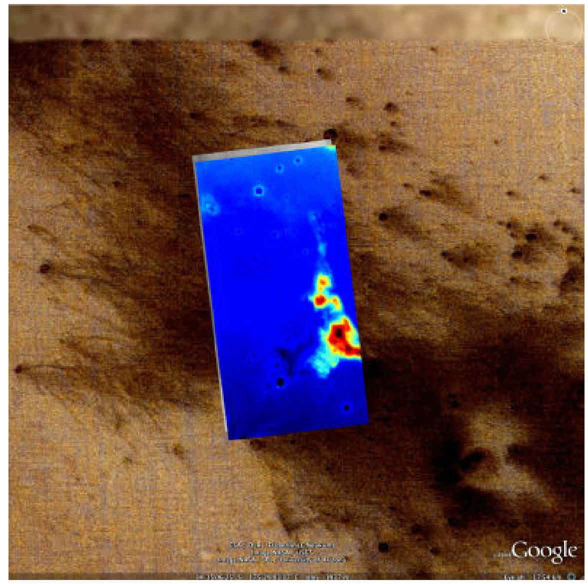
HiRISE observations PSP_001513_1655 and PSP_001777_1650 are on the floor of Gusev Crater and cover the area where the MER Spirit landed and has roved, including the Columbia Hills.

Commands

Download all 20 of the RED EDR `.IMG` files for each observation.



(a) `3D Rendering`



(b) `KML Screenshot`

Figure 8.1: Example output using HiRISE images PSP_001513_1655 and PSP_001777_1650 of the Columbia Hills.

```
ISIS 3> hiedr2mosaic.py PSP_001513_1655_RED*.IMG
ISIS 3> hiedr2mosaic.py PSP_001777_1650_RED*.IMG
ISIS 3> cam2map4stereo.py PSP_001777_1650_RED.mos_hijitreged.norm.cub \
                          PSP_001513_1655_RED.mos_hijitreged.norm.cub
ISIS 3> stereo PSP_001513_1655.map.cub \
               PSP_001777_1650.map.cub result/output
```

stereo.default

The stereo.default example file (appendix B) should apply well to HiRISE. Just set `alignment-method` to `none` if using map-projected imagery. If you are not using map-projected imagery, set `alignment-method` to `homography` or `affineepipolar`. The `corr-kernel` value can usually be safely reduced to 21 pixels to resolve finer detail and faster processing for images with good contrast.

8.3 Mars Reconnaissance Orbiter CTX

Context Camera (CTX) is a moderate camera to work with. Processing times for CTX can be pretty long when using Bayes EM subpixel refinement. Otherwise the disparity between images is relatively small, allowing efficient computation and a reasonable processing time.

8.3.1 North Terra Meridiani

In this example, we use map-projected images. Map-projecting the images is the most reliable way to align the images for correlation. However when possible, use non-map-projected images with the `alignment-method affineepipolar` option. This greatly reduces the time spent in triangulation. For all cases using linescan cameras, triangulation of map-projected images is 10x slower than non-map-projected images.

This example is distributed in the `examples/CTX` directory.

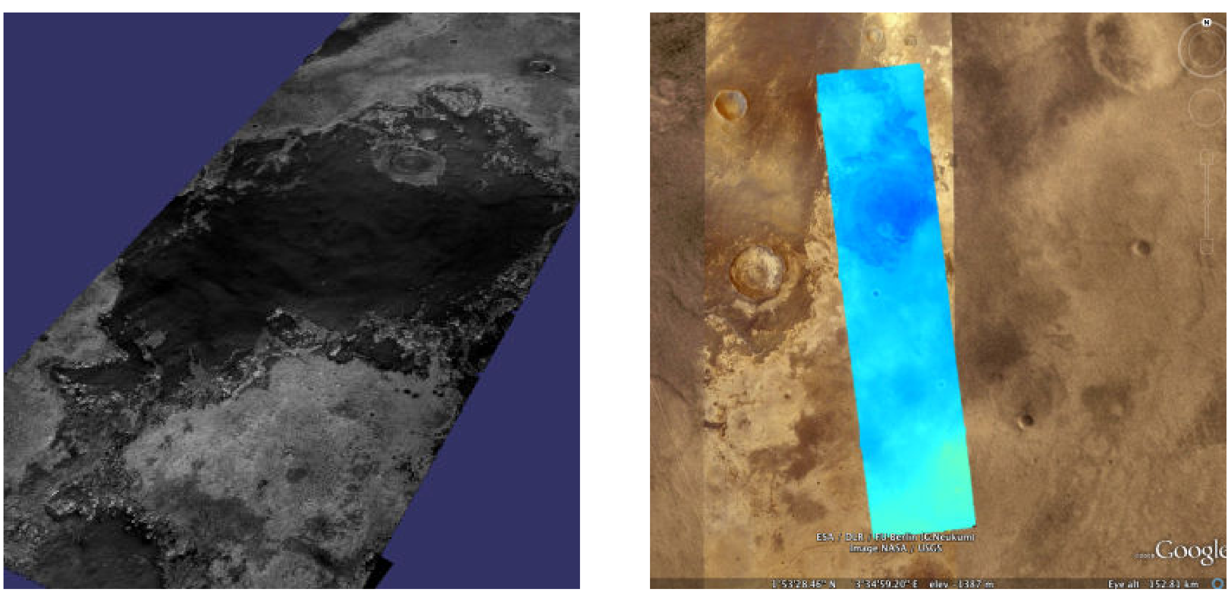
Commands

Download the CTX images P02_001981_1823_XI_02N356W.IMG and P03_002258_1817_XI_01N356W.IMG from the PDS.

```
ISIS 3> mroctx2isis from=P02_001981_1823_XI_02N356W.IMG to=P02_001981_1823.cub
ISIS 3> mroctx2isis from=P03_002258_1817_XI_01N356W.IMG to=P03_002258_1817.cub
ISIS 3> spiceinit from=P02_001981_1823.cub
ISIS 3> spiceinit from=P03_002258_1817.cub
ISIS 3> ctxcal from=P02_001981_1823.cub to=P02_001981_1823.cal.cub
ISIS 3> ctxcal from=P03_002258_1817.cub to=P03_002258_1817.cal.cub
  you can also optionally run ctxevenodd on the cal.cub files, if needed
ISIS 3> cam2map4stereo.py P02_001981_1823.cal.cub P03_002258_1817.cal.cub
ISIS 3> stereo P02_001981_1823.map.cub P03_002258_1817.map.cub results/out
```

stereo.default

The stereo.default example file (appendix B) works generally well with all CTX pairs. Just set `alignment-method` to `homography` or `affineepipolar`.



(a) `3D Rendering` (b) `KML Screenshot`

Figure 8.2: Example output possible with the CTX imager aboard MRO.

8.4 Mars Global Surveyor MOC-NA

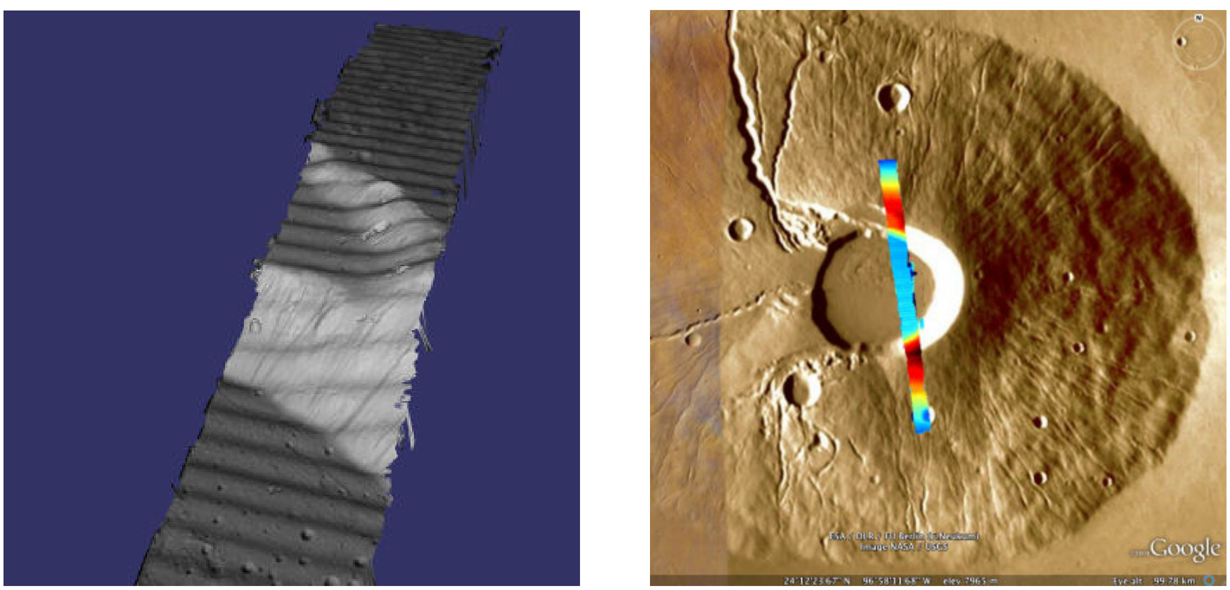
In the Stereo Pipeline Tutorial in Chapter 3, we showed you how to process a narrow angle MOC stereo pair that covered a portion of Hrad Vallis. In this section we will show you more examples, some of which exhibit a problem common to stereo pairs from linescan imagers: "spacecraft jitter" is caused by oscillations of the spacecraft due to the movement of other spacecraft hardware. All spacecraft wobble around to some degree but some are particularly susceptible.

Jitter causes wave-like distortions along the track of the satellite orbit in DEMs produced from linescan camera images. This effect can be very subtle or quite pronounced, so it is important to check your data products carefully for any sign of this type of artifact. The following examples will show the typical distortions created by this problem.

Note that the science teams of HiRISE and Lunar Reconnaissance Orbiter Camera (LROC) are actively working on detecting and correctly modeling jitter in their respective SPICE data. If they succeed in this, the distortions will still be present in the raw imagery, but the jitter will no longer produce ripple artifacts in the DEMs produced using ours or other stereo reconstruction software.

8.4.1 Ceraunius Tholus

Ceraunius Tholus is a volcano in northern Tharsis on Mars. It can be found at 23.96 N and 262.60 E. This DEM crosses the volcano's caldera.



(a) `3D Rendering` (b) `KML Screenshot`

Figure 8.3: Example output for MOC-NA of Ceraunius Tholus. Notice the presence of severe washboarding artifacts due to spacecraft "jitter."

Commands

Download the M08/06047 and R07/01361 images from the PDS.

```
ISIS 3> moc2isis f=M0806047.img t=M0806047.cub
```

```
ISIS 3> moc2isis f=R0701361.img t=R0701361.cub
ISIS 3> spiceinit from=M0806047.cub
ISIS 3> spiceinit from=R0701361.cub
ISIS 3> cam2map4stereo.py M0806047.cub R0701361.cub
ISIS 3> stereo M0806047.map.cub R0701361.map.cub result/output
```

stereo.default

The stereo.default example file (appendix B) works generally well with all MOC-NA pairs. Just set `alignment-method` to `none` when using map-projected imagery. If the images are not map-projected, use `homography` or `affineepipolar`.

8.5 Mars Exploration Rovers MER

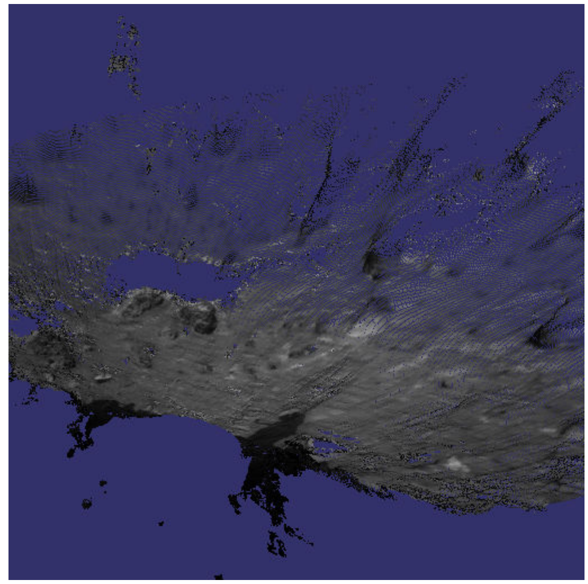
The MER rovers have several cameras on board and they all seem to have a stereo pair. With ASP you are able to process the PANCAM, NAVCAM, and HAZCAM camera imagery. ISIS has no telemetry or camera intrinsic supports for these images. That however is not a problem as their raw imagery contains the cameras' information in JPL's CAHV, CAHVOR, and CHAVORE formats.

These cameras are all variations of a simple pinhole camera model so they are processed with ASP in the `PINHOLE` session instead of the usual `ISIS`. ASP only supports creating of point clouds. *The *-PC.tif is a raw point cloud with the first 3 channels being XYZ in the rover site's coordinate frame.* We don't support the creation of DEMs from these images and that is left as an exercise for the user.

8.5.1 PANCAM, NAVCAM, HAZCAM

All of these cameras are processed the same way. I'll be showing 3D processing of the front hazard cams. The only new things in the pipeline is the new executable `mer2camera` along with the use of `alignment-method epipolar`. This example is also provided in the MER data example directory.

(a) `Rectified Input`



(b) `Output Point Cloud`

Figure 8.4: Example output possible with the front hazard cameras.

Commands

Download 2f194370083effap00p1214l0m1.img and 2f194370083effap00p1214r0m1.img from the PDS.

```
ISIS 3> mer2camera 2f194370083effap00p1214l0m1.img
ISIS 3> mer2camera 2f194370083effap00p1214r0m1.img
ISIS 3> stereo 2f194370083effap00p1214l0m1.img 2f194370083effap00p1214r0m1.img \
              2f194370083effap00p1214l0m1.cahvore 2f194370083effap00p1214r0m1.cahvore \
              fh01/fh01
```

stereo.default

The default stereo settings will work but change the following options. The universe option filters out points that are not triangulated well because they are too close *robot's hardware* or are extremely far away.

additional settings for MER

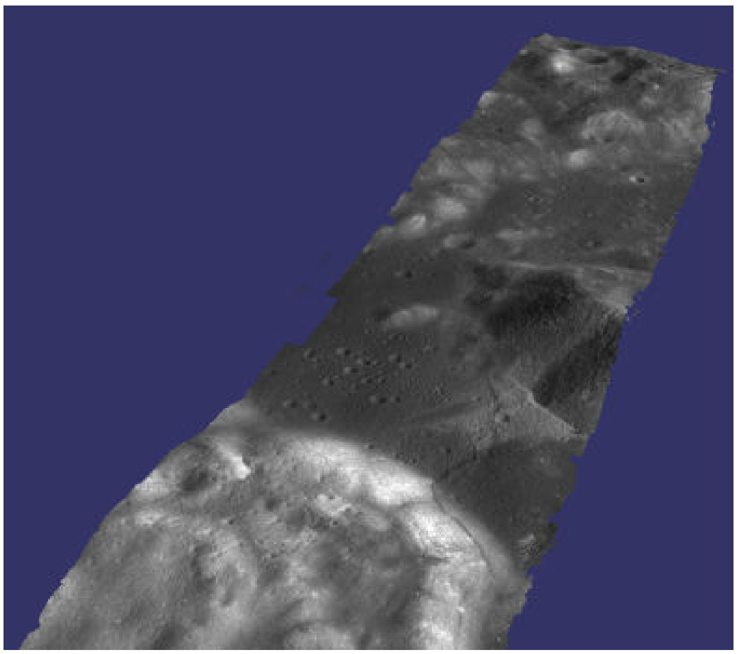
```
alignment-method epipolar
force-use-entire-range

# This deletes points that are too far away
# from the camera to truly triangulate.
universe-center Camera
near-universe-radius 0.7
far-universe-radius 80.0
```

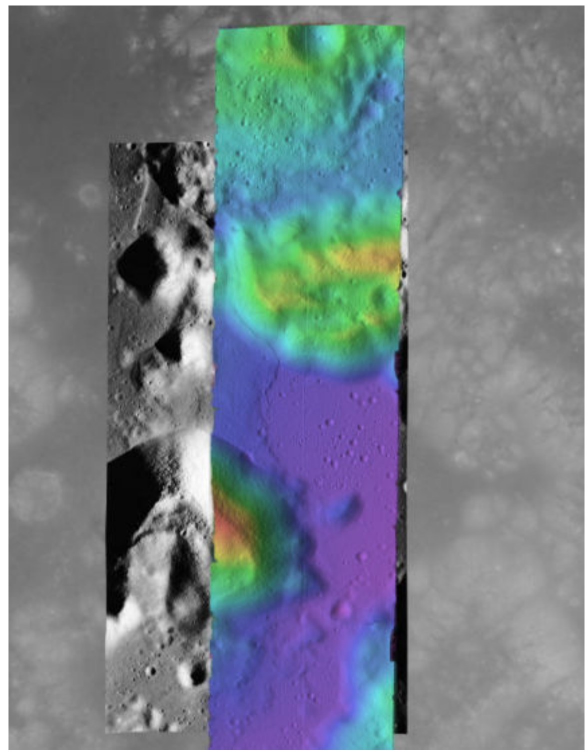
8.6 Lunar Reconnaissance Orbiter LROC NAC

8.6.1 Lee-Lincoln Scarp

This stereo pair covers the Taurus-Littrow valley on the Moon where, on December 11, 1972, the astronauts of Apollo 17 landed. However, this stereo pair does not contain the landing site. It is slightly west; focusing on the Lee-Lincoln scarp that is on North Massif. The scarp is an 80 m high feature that is the only visible sign of a deep fault.



(a) `3D Rendering`



(b) `KML Screenshot`

Figure 8.5: Example output possible with a LROC NA stereo pair, using both CCDs from each observation courtesy of the lronac2mosaic.py tool.

Commands

Download the EDRs for the left and right CCDs for observations M104318871 and M104318871. Alternatively you can search by original IDs of 2DB8 and 4C86 in the PDS.

All ISIS preprocessing of the EDRs is performed via the `lronac2mosaic.py` command. This runs `lronac2isis`, `lronaccal`, `lronacecho`, `spiceinit`, `noproj`, and `handmos` to create a stitched unprojected image for a single observation. In this example we don't map-project the images as ASP can usually get good results. More aggressive terrain might require an additional `cam2map4stereo.py` step.

```
ISIS 3> lronac2mosaic.py M104318871LE.img M104318871RE.img
ISIS 3> lronac2mosaic.py M104311715LE.img M104311715RE.img
ISIS 3> stereo M104318871LE*.mosaic.norm.cub M104311715LE*.mosaic.norm.cub \
          result/output --alignment-method affineepipolar
```

stereo.default

The defaults work generally well with LRO-NAC pairs, so you don't need to provide a stereo.default file. Map-projecting is optional. When map-projecting the images use `alignment-method none`, otherwise use `alignment-method affineepipolar`. Better map-project results can be achieved by projecting on a higher resolution elevation source like the WAC DTM. This is achieved using the ISIS command `demprep` and attaching to cube files via `spiceinit`'s SHAPE and MODEL options.

8.7 Apollo 15 Metric Camera Images

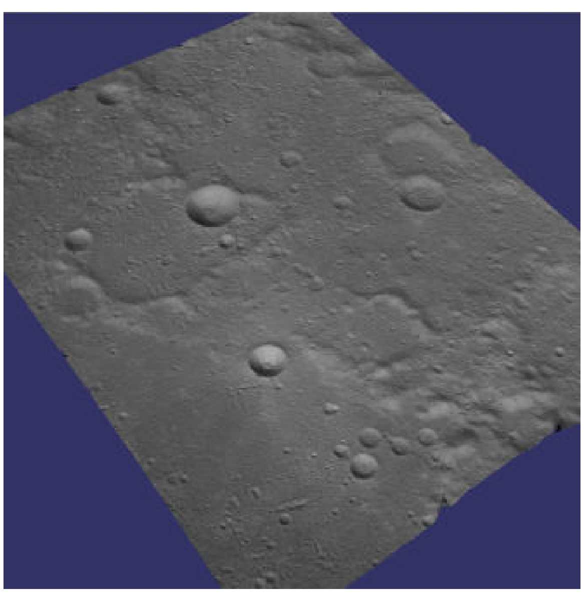
Apollo Metric images were all taken at regular intervals, which means that the same `stereo.default` can be used for all sequential pairs of images. Apollo Metric images are ideal for stereo processing. They produce consistent, excellent results.

The scans performed by ASU are sufficiently detailed to exhibit film grain at the highest resolution. The amount of noise at the full resolution is not helpful for the correlator, so we recommend subsampling the images by a factor of 4.

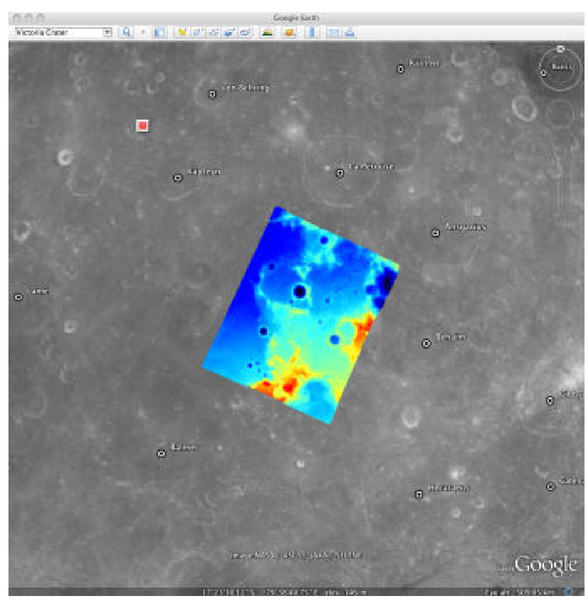
Currently the tools to ingest Apollo TIFFs into ISIS are not available, but these images should soon be released into the PDS for general public usage.

8.7.1 Ansgarius C

Ansgarius C is a small crater on the west edge of the far side of the Moon near the equator. It is east of Kapteyn A and B.



(a) `3D Rendering`



(b) `KML Screenshot`

Figure 8.6: Example output possible with Apollo Metric frames AS15-M-2380 and AS15-M-2381.

Commands

Process Apollo TIFF files into ISIS.

```
ISIS 3> reduce from=AS15-M-2380.cub to=sub4-AS15-M-2380.cub sscale=4 lscale=4
ISIS 3> reduce from=AS15-M-2381.cub to=sub4-AS15-M-2381.cub sscale=4 lscale=4
ISIS 3> spiceinit from=sub4-AS15-M-2380.cub
ISIS 3> spiceinit from=sub4-AS15-M-2381.cub
ISIS 3> stereo sub4-AS15-M-2380.cub sub4-AS15-M-2381.cub result/output
```

stereo.default

The stereo.default example file (appendix B) works generally well with all Apollo pairs. Just set `alignment-method` to `homography` or `affineepipolar`.

8.8 Cassini ISS NAC

This is a proof of concept showing the strength of building the Stereo Pipeline on top of ISIS. Support for processing ISS NAC stereo pairs was not a goal during our design of the software, but the fact that a camera model exists in ISIS means that it too can be processed by the Stereo Pipeline.

Identifying stereo pairs from spacecraft that do not orbit their target is a challenge. We have found that one usually has to settle with images that are not ideal: different lighting, little perspective change, and little or no stereo parallax. So far we have had little success with Cassini's data, but nonetheless we provide this example as a potential starting point.

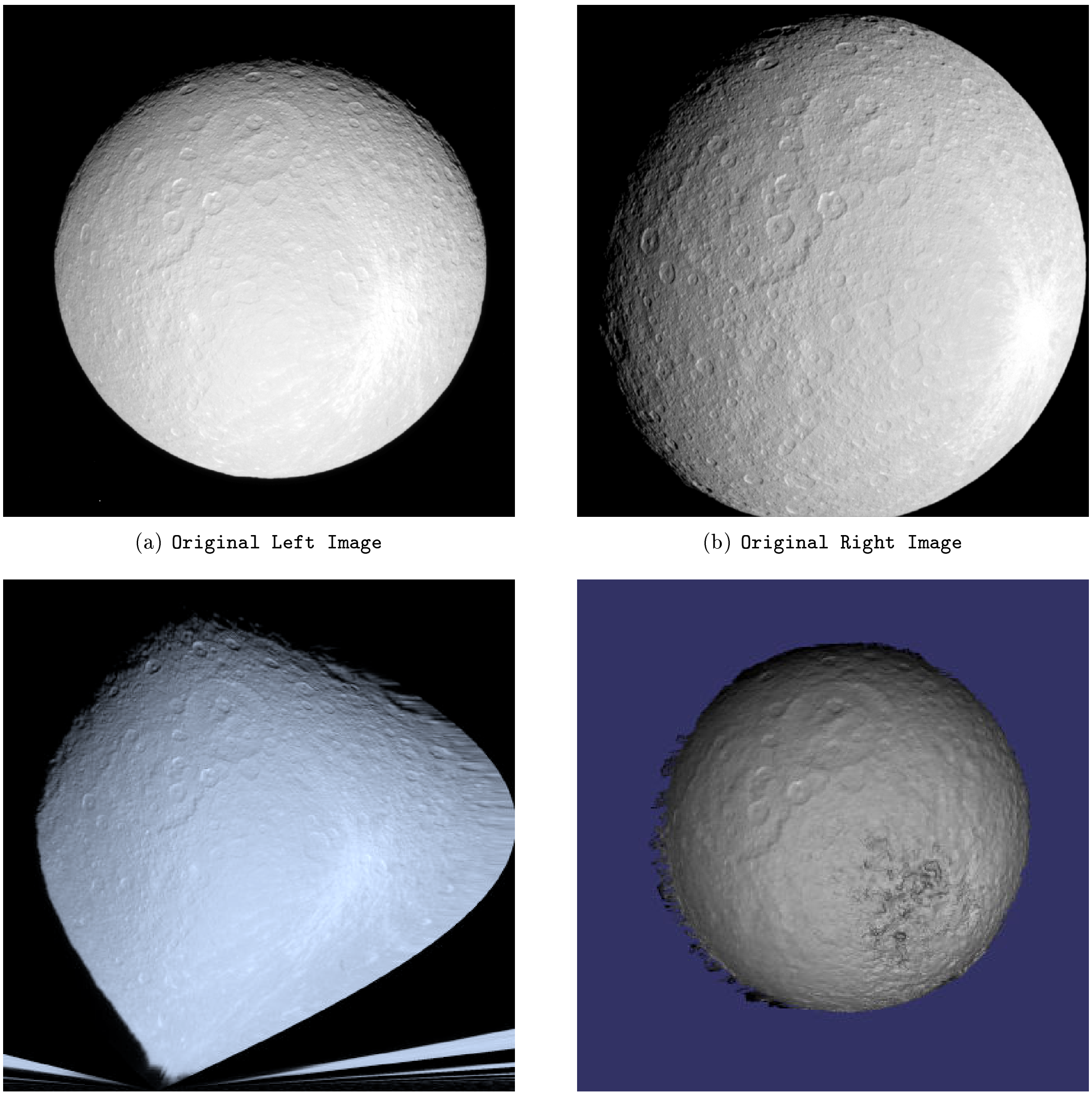
8.8.1 Rhea

Rhea is the second largest moon of Saturn and is roughly a third the size of our own Moon. This example shows, at the top right of both images, a giant impact basin named Tirawa that is 220 miles across. The bright white area south of Tirawa is ejecta from a new crater. The lack of texture in this area poses a challenge for our correlator. The results are just barely useful: the Tirawa impact can barely be made out in the 3D data while the new crater and ejecta become only noise.

Commands

Download the N1511700120_1.IMG and W1567133629_1.IMG images and their label (.LBL) files from the PDS.

```
ISIS 3> ciss2isis f=N1511700120_1.LBL t=N1511700120_1.cub
ISIS 3> ciss2isis f=W1567133629_1.LBL t=W1567133629_1.cub
ISIS 3> cisscal from=N1511700120_1.cub to=N1511700120_1.lev1.cub
ISIS 3> cisscal from=W1567133629_1.cub to=W1567133629_1.lev1.cub
ISIS 3> fillgap from=W1567133629_1.lev1.cub to=W1567133629_1.fill.cub %Only one image
                                                                      %exhibits the problem
ISIS 3> cubenorm from=N1511700120_1.lev1.cub to=N1511700120_1.norm.cub
ISIS 3> cubenorm from=W1567133629_1.fill.cub to=W1567133629_1.norm.cub
ISIS 3> spiceinit from=N1511700120_1.norm.cub
ISIS 3> spiceinit from=W1567133629_1.norm.cub
ISIS 3> cam2map from=N1511700120_1.norm.cub to=N1511700120_1.map.cub
ISIS 3> cam2map from=W1567133629_1.norm.cub map=N1511700120_1.map.cub \
ISIS 3>          to=W1567133629_1.map.cub matchmap=true
ISIS 3> stereo N1511700120_1.map.equ.cub W1567133629_1.map.equ.cub result/rhea
```



(a) `Original Left Image` (b) `Original Right Image`

(c) `Map-Projected Left` (d) `3D Rendering`

Figure 8.7: Example output of what is possible with Cassini's ISS NAC

stereo.default

stereo.default for Cassini ISS

```
### PREPROCESSING
alignment-method none
force-use-entire-range
individually-normalize

### CORRELATION
prefilter-mode 2
prefilter-kernel-width 1.5

cost-mode 2

corr-kernel 25 25
corr-search -55 -2 -5 10

subpixel-mode 3
subpixel-kernel 21 21

### FILTERING
rm-half-kernel 5 5
rm-min-matches 60 # Units = percent
rm-threshold 3
rm-cleanup-passes 1
```

8.9 Digital Globe Imagery

Processing of Digital Globe images is described extensively in the tutorial in chapter 4.

8.10 GeoEye and Astrium Imagery / RPC Imagery

GeoEye provides imagery from Ikonos and the two GeoEye satellites. Astrium provides imagery from SPOT and Pleiades satellites. Both companies provide only Rational Polynomial Camera (RPC) models. RPC represents four 20-element polynomials that map geodetic coordinates to image coordinates. Since they are easy to implement, RPC represents a universal camera model and can be had from many imaging providers; Digital Globe also provides them. The only downside is that it has less precision in our opinion compared to the linear camera model provided by Digital Globe. For GeoEye and Astrium, the only option is using RPC.

Our RPC read driver is GDAL. If the command `gdalinfo` can identify the RPC information inside the headers of your files, ASP will likely be able to see it as well. This means that sometimes we can get away with only providing a left and right image, with no extra files containing camera information. This is specifically the case for GeoEye.

You can download an example stereo pair from GeoEye's website at [11]. When we accessed the site, we downloaded a GeoEye-1 image of Hobart, Australia. As previously stated in the Digital Globe section, these types of images are not ideal for ASP. This is both a forest and a urban area which makes correlation difficult. ASP was designed more for modeling bare rock and ice. Any results we produce in other environments is a bonus but is not our objective.

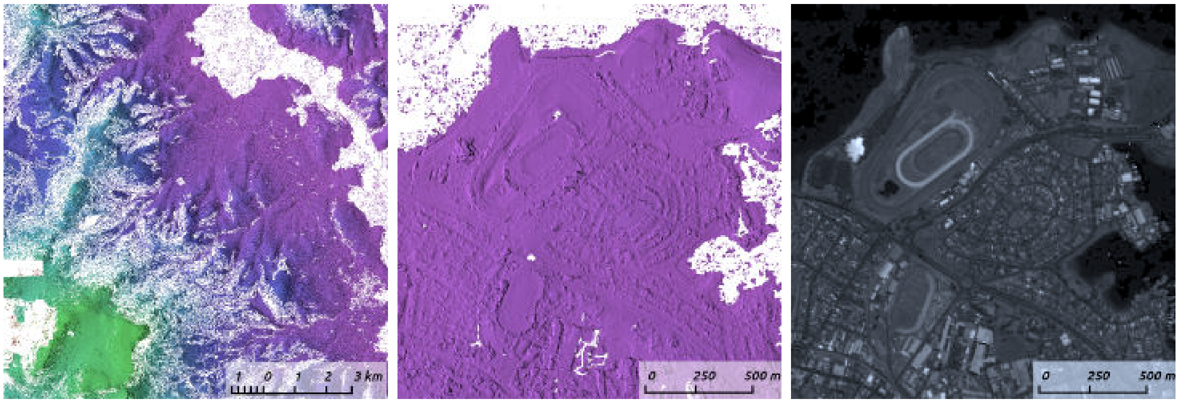


Figure 8.8: Example colorized height map and ortho image output.

Commands

```
> stereo -t rpc po_312012_pan_0000000.tif po_312012_pan_0010000.tif geoeye/geoeye
```

In case the image files do not contain the RPC models, separate XML files having this information need to be provided, as done for Digital Globe images (section 4.1).

Currently, stereo using RPC camera models cannot be performed if the input images are map-projected, as it is possible with Digital Globe images with linear camera models (section 4.2).

stereo.default

The stereo.default example file (appendix B) works generally well with all GeoEye pairs. Just set `alignment-method` to `affineepipolar` or `homography`.

8.11 Dawn (FC) Framing Camera

This is a NASA mission to visit two of the largest objects in the asteroid belt, Vesta and Ceres. The framing camera on board Dawn is quite small and packs only a resolution of 1024x1024 pixels. This means processing time is extremely short. To its benefit, it seems that the mission planners leave the framing camera on taking shots quite rapidly. On a single pass, they seem to usually take a chain of FC images that have a high overlap percentage. This opens the idea of using ASP to process not only the sequential pairs, but also the wider baseline shots. Then someone could potentially average all the DEMs together to create a more robust data product.

For this example, we downloaded the images

`FC21A0010191_11286212239F1T.IMG` and `FC21A0010192_11286212639F1T.IMG`

which show the Cornelia crater. We found these images by looking at the popular anaglyph shown on the Planetary Science Blog [16].

Commands

First you must download the Dawn FC images from PDS.

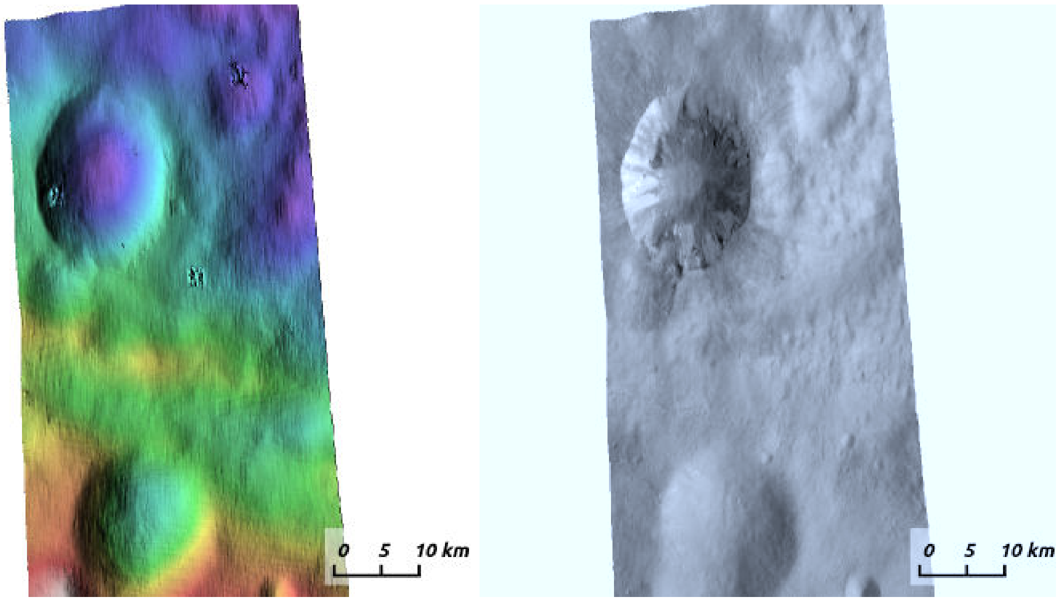


Figure 8.9: Example colorized height map and ortho image output.

```
ISIS3 > dawnfc2isis from=FC21A0010191_11286212239F1T.IMG \
                    to=FC21A0010191_11286212239F1T.cub
ISIS3 > dawnfc2isis from=FC21A0010192_11286212639F1T.IMG \
                    to=FC21A0010192_11286212639F1T.cub
ISIS3 > spiceinit from=FC21A0010191_11286212239F1T.cub
ISIS3 > spiceinit from=FC21A0010192_11286212639F1T.cub
ISIS3 > stereo FC21A0010191_11286212239F1T.cub \
               FC21A0010192_11286212639F1T.cub stereo/stereo
ISIS3 > point2dem stereo-PC.tif --orthoimage stereo-L.tif \
--t_srs "+proj=eqc +lat_ts=-11.5 +a=280000 +b=229000 +units=m"
```

stereo.default

The stereo.default example file (appendix B) works well for this stereo pair. Just set `alignment-method` to `affineepipolar` or `homography`.

Part III

Appendices

Appendix A

Tools

This chapter provides a overview of the various tools that are provided as part of the Ames Stereo Pipeline, and a summary of their command line options.

A.1 stereo

The **stereo** program is the primary tool of the Ames Stereo Pipeline. It takes a stereo pair of images that overlap and creates an output point cloud image that can be processed into a visualizable mesh or a DEM using **point2mesh** (section A.5) and **point2dem** (section A.4), respectively.

Usage:

```
ISIS 3> stereo [options] <images> [<cameras>] output_file_prefix
```

Example (for ISIS):

```
stereo file1.cub file2.cub results/run
```

For ISIS, a .cub file has both image and camera information, as such no separate camera files are specified.

Example (for Digital Globe Earth images):

```
stereo file1.tif file2.tif file1.xml file2.xml results/run
```

Multiple input images are also supported (section 5.1.5).

This tool is is primarily designed to process USGS ISIS `.cub` files and Digital Globe data. However, Stereo Pipeline does have the capability to process other types of stereo image pairs (e.g., image files with a CAHVOR camera model from the NASA MER rovers). If you would like to experiment with these features, please contact us for more information.

The ***output_file_prefix*** is prepended to all output data files. For example, setting ***output_file_prefix*** to '`out`' will yield files with names like `out-L.tif` and `out-PC.tif`. To keep the Stereo Pipeline results organized in sub-directories, we recommend using an output prefix like '`results-10-12-09/out`' for ***output_file_prefix***. The **stereo** program will create a directory called `results-10-12-09/` and place files named `out-L.tif`, `out-PC.tif`, etc. in that directory.

Table A.1: Command-line options for stereo

Option	Description
`--help|-h`	Display the help message
`--threads` *integer(=0)*	Set the number of threads to use. 0 means use as many threads as there are cores.
`--session-type|-t pinhole|isis|dg|rpc`	Select the stereo session type to use for processing. Usually the program can select this automatically by the file extension.
`--stereo-file|-s` *filename(=./stereo.default)*	Define the stereo.default file to use
`--left-image-crop-win` *xoff yoff xsize ysize*	Do stereo in a sub-region of the left image [default: use the entire image].
`--entry-point|-e integer(=0 to 4)`	Stereo Pipeline entry point (start at this stage).
`--stop-point|-e integer(=1 to 5)`	Stereo Pipeline stop point (stop at the stage *right before* this value).
`--corr-seed-mode integer(=0 to 3)`	Correlation seed strategy (section B.2).

More information about the stereo.default configuration file can be found in Appendix B on page 99. `stereo` creates a set of intermediate files, they are described in Appendix C on page 105.

A.1.1 Entry Points

The `stereo -e` *`number`* option can be used to restart a `stereo` job partway through the stereo correlation process. Restarting can be useful when debugging while iterating on `stereo.default` settings.

Stage 0 (Preprocessing) normalizes the two images and aligns them by locating interest points and matching them in both images. The program is designed to reject outlying interest points. This stage writes out the pre-aligned images and the image masks.

Stage 1 (Disparity Map Initialization) performs pyramid correlation and builds a rough disparity map that is used to seed the sub-pixel refinement phase.

Stage 2 (Sub-pixel Refinement) performs sub-pixel correlation that refines the disparity map.

Stage 3 (Outlier Rejection and Hole Filling) performs filtering of the disparity map and (optionally) fills in holes using an inpainting algorithm. This phase also creates a “good pixel” map.

Stage 4 (Triangulation) generates a 3D point cloud from the disparity map.

A.1.2 Decomposition of Stereo

The `stereo` executable is a python script that makes calls to separate C++ executables for each entry point.

Stage 0 (Preprocessing) calls `stereo_pprc`. Multi-threaded.

Stage 1 (Disparity Map Initialization) calls `stereo_corr`. Multi-threaded.

Stage 2 (Sub-pixel Refinement) class `stereo_rfne`. Multi-threaded.

Stage 3 (Outlier Rejection and Hole Filling) calls `stereo_fltr`. Multi-threaded.

Stage 4 (Triangulation) calls `stereo_tri`. Multi-threaded, except for ISIS input data.

All of the sub-programs have the same interface as `stereo`. Users processing a large number of stereo pairs on a cluster may find it advantageous to call these executables in their own manner. An example would be to run stages 0-3 in order for each stereo pair. Then run several sessions of `stereo_tri` since it is single-threaded for ISIS.

It is important to note that each of the C++ stereo executables invoked by `stereo` have their own command-line options. Those options can be passed to `stereo` which will in turn pass them to the appropriate executable. By invoking each executable with no options, it will display the list of options it accepts.

As explained in more detail in section 5.1.2, each such option has the same syntax as used in `stereo.default`, while being prepended by a double hyphen (`--`). A command line option takes precedence over the same option specified in `stereo.default`. Chapter B documents all options for the individual sub-programs.

A.2 parallel_stereo

The `parallel_stereo` program is a modification of `stereo` designed to distribute the stereo processing over multiple computing nodes. It uses GNU Parallel to manage the jobs, a tool which is distributed along with Stereo Pipeline. It expects that all nodes can connect to each other using ssh without password. `parallel_stereo` can also be useful when processing extraterrestial data on a single computer. This is because ISIS camera models are restricted to a single thread, but `parallel_stereo` can run multiple processes in parallel to reduce computation times.

At the simplest, `parallel_stereo` can be invoked exactly like stereo, with the addition of the list of nodes to use (if using multiple nodes).

```
parallel_stereo --nodes-list machines.txt <other stereo options>
```

If your jobs are launched on a cluster or supercomputer, the name of the file containing the list of nodes may exist as an environmental variable. For example, on NASA's Pleiades Supercomputer, which uses the Portable Batch System (PBS), the list of nodes can be retrieved as $PBS_NODEFILE.

It is important to note that when invoking this tool only stages 1, 2, and 4 of stereo (section A.1.2) are spread over multiple machines, with stages 0 and 3 using just one node, as they require global knowledge of the data. In addition, not all stages of stereo benefit equally from parallelization. Most likely to gain are stages 1 and 2 (correlation and refinement) which are the most computationally expensive.

For these reasons, while `parallel_stereo` can be called to do all stages of stereo generation from start to finish in one command, it may be more resource-efficient to invoke it using a single node for stages 0 and 3, many nodes for stages 1 and 2, and just a handful of nodes for stage 4 (triangulation). For example, to invoke the tool only for stage 2, one uses the options:

```
--entry-point 2 --stop-point 3
```

`parallel_stereo` accepts the following options (any additional options given to it will be passed to the stereo executables for each stage).

Table A.2: Command-line options for parallel_stereo

Options	Description

`--help|-h`	Display the help message.
`--nodes-list` *filename*	The list of computing nodes, one per line. If not provided, run on the local machine.
`--entry-point|-e integer(=0 to 4)`	Stereo Pipeline entry point (start at this stage).
`--stop-point|-e integer(=1 to 5)`	Stereo Pipeline stop point (stop at the stage *right before* this value).
`--corr-seed-mode integer(=0 to 3)`	Correlation seed strategy (section B.2).

A.2.1 Advanced usage

The `parallel_stereo` tool tries to take advantage of its inside knowledge of the individual `stereo` sub-programs to decide how many threads and processes to use at each stage, and by default, it it will try to use all nodes to the fullest.

The advanced user can try to gain finer-level control of the tool, as described below. This may not necessarily result in improved performance compared to using the default settings.

As an example of using the advanced options, assume that we would like to launch the refinement and filtering steps only (stages 2 and 3). We will distribute the refinement over a number of nodes, using 4 processes on each node, with each process creating 16 threads. For the filtering stage, which is done in one process on one machine, we want to use 32 threads. The appropriate command is then:

```
parallel_stereo --nodes-list machines.txt --processes 4 --threads-multiprocess 16 \
 --threads-singleprocess 32 --entry-point 2 --stop-point 4 <other stereo options>
```

To better take advantage of these options, the user should know the following. `parallel_stereo` starts a process for every image block, whose size is by default 2048×2048 (`job-size-w` by `job-size-h`). On such a block, the correlation, and subpixel refinement stages will use at most 4 and 64 threads respectively (1 and 16 threads for each 1024×1024 tile). Triangulation will use at most 64 threads as well, except for ISIS cameras, when it is single-threaded due to the limitations of ISIS (we account for the latter when the number of threads and processes are decided automatically, but not when these advanced options are used).

Table A.3: Advanced options for parallel_stereo

Options	Description
`--job-size-w` *integer(=2048)*	Pixel width of input image tile for a single process.
`--job-size-h` *integer(=2048)*	Pixel height of input image tile for a single process.
`--processes` *integer*	The number of processes to use per node.
`--threads-multiprocess` *integer*	The number of threads to use per process.
`--threads-singleprocess` *integer*	The number of threads to use when running a single process (for pre-processing and filtering).

A.3 bundle_adjust

The `bundle_adjust` program performs bundle adjustment on a given set of images and cameras. An introduction to bundle adjustment can be found in chapter 7, with an example of how to use this program in section 7.2.

This tool can use several algorithms for bundle adjustment. The default is to use Google's Ceres Solver (http://ceres-solver.org/).

Usage:

```
bundle_adjust <images> <cameras> <optional ground control points> \
  -o <output prefix> [options]
```

Example (for ISIS):

```
bundle_adjust file1.cub file2.cub file3.cub -o results/run
```

Example (for Digital Globe Earth data, using ground control points):

```
bundle_adjust file1.tif file2.tif file1.xml file2.xml gcp_file.gcp \
  --datum WGS_1984 -o results/run
```

The `stereo` program can then be told to use the adjusted cameras via the option `--bundle-adjust-prefix`.

Table A.4: Command-line options for bundle_adjust

Option	Description
`--help|-h`	Display the help message.
`--output-prefix|-o` *filename*	Prefix for output filenames.
`--bundle-adjuster` *string [default: Ceres]*	Choose a solver from: Ceres, RobustSparse, RobustRef, Sparse, Ref.
`--cost-function` *string [default: Cauchy]*	Choose a cost function from: Cauchy, PseudoHuber, Huber, L1, L2.
`--robust-threshold` *double(=0.5)*	Set the threshold for robust cost functions.
`--datum` *string*	Use this datum (needed only if ground control points are used). Options: WGS_1984, D_MOON (radius is assumed to be 1,737,400 meters), D_MARS (radius is assumed to be 3,396,190 meters), etc.
`--semi-major-axis` *double*	Explicitly set the datum semi-major axis in meters (needed only if ground control points are used).
`--semi-minor-axis` *double*	Explicitly set the datum semi-minor axis in meters (needed only if ground control points are used).
`--session-type|-t pinhole|isis|dg|rpc`	Select the stereo session type to use for processing. Usually the program can select this automatically by the file extension.
`--min-matches` *integer(=30)*	Set the minimum number of matches between images that will be considered.

`--max-iterations` *integer(=100)*	Set the maximum number of iterations.
`--overlap-limit` *integer(=3)*	Limit the number of subsequent images to search for matches to the current image to this value.
`--camera-weight` *double(=1.0)*	The weight to give to the constraint that the camera positions/orientations stay close to the original values (only for the Ceres solver).
`--lambda` *double*	Set the initial value of the LM parameter lambda (ignored for the Ceres solver).
`--threads` *integer(=0)*	Set the number threads to use. 0 means use the default defined in the program or in the .vwrc file.
`--report-level|-r` *integer=(10)*	Use a value >= 20 to get increasingly more verbose output.

The `bundle_adjust` program will save the obtained adjustments (rotation and translation) for each camera in plain text files whose names start with the specified output prefix. This prefix can then be passed to `stereo` via the option `--bundle-adjust-prefix`.

A number of plain-text files containing ground control points can be passed as input to `bundle_adjust`. Such a file must end with a .gcp extension, and contain one ground control point per line. Each line must have the following fields:

- ground control point id (integer)
- latitude (in degrees)
- longitude (in degrees)
- height above datum (in meters), with the datum itself specified separately
- x, y, z standard deviations (three positive floating point numbers, smaller values suggest more reliable measurements)

On the same line, for each image in which the ground control point is visible there should be:

- image file name
- column index in image (float)
- row index in image (float)
- column and row standard deviations (two positive floating point numbers, smaller values suggest more reliable measurements)

The fields can be separated by spaces or commas. Here is a sample representation of a ground control point measurement:

```
5 23.7 160.1 427.1 1.0 1.0 1.0 image1.tif 124.5 19.7 1.0 1.0 image2.tif 254.3 73.9 1.0 1.0
```

A.4 point2dem

The `point2dem` program produces a GeoTIFF terrain model and/or an orthographic image from a set of point clouds. The clouds can be created by the `stereo` command, or be in LAS or CSV format.

Example:

```
point2dem output-prefix-PC.tif -o stereo/filename -r moon \
    --nodata-value -10000 -n
```

This produces a digital elevation model that has been referenced to the lunar spheroid of 1737.4 km. Pixels with no data will be set to a value of -10000, and the resulting DEM will be saved in a simple cylindrical map-projection. The resulting DEM is stored by default as a one channel, 32-bit floating point GeoTIFF file.

The `-n` option creates an 8-bit, normalized version of the DEM that can be easily loaded into a standard image viewing application for debugging.

Another example:

```
point2dem output-prefix-PC.tif -o stereo/filename -r moon \
    --orthoimage output-prefix-L.tif
```

This command takes the left input image and orthographically projects it onto the 3D terrain produced by the Stereo Pipeline. The resulting `*-DRG.tif` file will be saved as a GeoTIFF image in a simple cylindrical map-projection.

Multiple point clouds can be passed as inputs, to be combined into a single DEM. If it is desired to use the `--orthoimage` option as above, the clouds need to be specified first, followed by the `L.tif` images. Here is an example, which combines together LAS and CSV point clouds together with an output file from `stereo`:

```
point2dem in1.las in2.csv output-prefix-PC.tif -o combined \
  --dem-spacing 0.001 --nodata-value -32768
```

A.4.1 Comparing with MOLA Data

When comparing the output of `point2dem` to laser altimeter data, like MOLA, it is important to understand the different kinds of data that are being discussed. By default, `point2dem` returns planetary radius values in meters. These are often large numbers that are difficult to deal with. If you use the `-r mars` option, the output terrain model will be in meters of elevation with reference to the IAU reference spheroid for Mars: 3,396,190 m. So if a post would have a radius value of 3,396,195 m, in the model returned with the `-r mars` option, that pixel would just be 5 m.

You may want to compare the output to MOLA data. MOLA data is released in three 'flavors,' namely: Topography, Radius, and Areoid. The MOLA Topography data product that most people use is just the MOLA Radius product with the MOLA Areoid product subtracted. Additionally, it is important to note that all of these data products have a reference value subtracted from them. The MOLA reference value is NOT the IAU reference value, but 3,396,000 m.

In order to compare with the MOLA data, you can do one of two different things. You could operate purely in radius space, and have `point2dem` create radius values that are directly comparable to the MOLA Radius data. You can do this by having `point2dem` subtract the MOLA reference value by setting `--semi-major-axis 3396000` and `--semi-minor-axis 3396000`.

To get values that are directly comparable to MOLA Topography data, you'll need to run `point2dem` with the option `-r mars`, then run the ASP tool `dem_geoid` (section A.7). This program will convert the DEM height values from being relative to the IAU reference spheroid to being relative to the MOLA Areoid.

A.4.2 Post Spacing

Recall that `stereo` creates a point cloud file as its output and that you need to use `point2dem` on to create a GeoTIFF that you can use in other tools. The point cloud file is the result of taking the image-to-image matches (which were created from the kernel sizes you specified, and the subpixel versions of the same, if used) and projecting them out into space from the cameras, and arriving at a point in real world coordinates. Since `stereo` does this for every pixel in the input images, the *default* value that `point2dem` uses (if you don't specify anything explicitly) is the input image scale, because there's an 'answer' in the point cloud file for each pixel in the original image.

However, as you may suspect, this is probably not the best value to use because there really isn't that much 'information' in the data. The true 'resolution' of the output model is dependent on a whole bunch of things (like the kernel sizes you choose to use) but also can vary from place to place in the image depending on the texture.

The general 'rule of thumb' is to produce a terrain model that has a post spacing of about 3x the input image ground scale. This is based on the fact that it is nearly impossible to uniquely identify a single pixel correspondence between two images, but a 3x3 patch of pixels provides improved matching reliability. As you go to numerically larger post-spacings on output, you're averaging more point data (that is probably spatially correlated anyway) together.

So you can either use the `--dem-spacing` argument to `point2dem` to do that directly, or you can use your favorite averaging algorithm to reduce the `point2dem`-created model down to the scale you want.

If you attempt to derive science results from an ASP-produced terrain model with the default DEM spacing, expect serious questions from reviewers.

A.4.3 Using with LAS or CSV Clouds

The `point2dem` program can take as inputs point clouds in LAS and CSV formats. These differ from point clouds created by stereo by being, in general, not uniformly distributed. It is suggested that the user pick carefully the output resolution for such files (`--dem-spacing`). If the output DEM turns out to be sparse, the spacing could be increased, or one could experiment with increasing the value of `--search-radius-factor`, which will fill in small gaps in the output DEM by searching further for points in the input clouds.

It is expected that the input LAS files have spatial reference information such as WKT data. Otherwise it is assumed that the points are raw x, y, z values in meters in reference to the planet center.

Unless the output projection is explicitly set when invoking `point2dem` the one from the first LAS file will be used.

For LAS or CSV clouds it is not possible to generate intersection error maps or ortho images.

For CSV clouds the option `--csv-format` must be set.

Table A.5: Command-line options for point2dem

Options	Description
`--nodata-value` *float(=min-z)*	Explicitly set the default missing pixel value. By default, the minimum z value in the model is used.
`--use-alpha`	Create images that have an alpha channel.
`--normalized|-n`	Also write a normalized version of the DEM (for debugging).
`--orthoimage`	Write an orthoimage based on the texture files passed in as inputs (after the point clouds).

`--errorimage`	Write an additional image whose values represent the triangulation error in meters.
`--output-prefix|-o` *output-prefix*	Specify the output prefix.
`--output-filetype|-t` *type(=tif)*	Specify the output file type.
`--x-offset` *float(=0)*	Add a horizontal offset to the DEM.
`--y-offset` *float(=0)*	Add a horizontal offset to the DEM.
`--z-offset` *float(=0)*	Add a vertical offset to the DEM.
`--rotation-order` *order(=xyz)*	Set the order of an Euler angle rotation applied to the 3D points prior to DEM rasterization.
`--phi-rotation` *float(=0)*	Set a rotation angle phi.
`--omega-rotation` *float(=0)*	Set a rotation angle omega.
`--kappa-rotation` *float(=0)*	Set a rotation angle kappa.
`--t_srs` *string*	Specify the projection (PROJ.4 string).
`--reference-spheroid|-r earth|moon|mars`	Set the reference spheroid. This will override manually set datum information.
`--semi-major-axis` *float(=0)*	Explicitly set the datum semi-major axis in meters.
`--semi-minor-axis` *float(=0)*	Explicitly set the datum semi-minor axis in meters.
`--sinusoidal`	Save using a sinusoidal projection.
`--mercator`	Save using a Mercator projection.
`--transverse-mercator`	Save using transverse Mercator projection.
`--orthographic`	Save using an orthographic projection.
`--stereographic`	Save using a stereographic projection.
`--lambert-azimuthal`	Save using a Lambert azimuthal projection.
`--utm` *zone*	Save using a UTM projection with the given zone.
`--proj-lat` *float*	The center of projection latitude (if applicable).
`--proj-lon` *float*	The center of projection longitude (if applicable).
`--proj-scale` *float*	The projection scale (if applicable).
`--dem-spacing|-s` *float(=0)*	Set the output DEM resolution (in target georeferenced units per pixel). If not specified, it will be computed automatically (except for LAS and CSV files).
`--search-radius-factor` *float(=0)*	Multiply this factor by `dem-spacing` to get the search radius. The DEM height at a given grid point is obtained as a weighted average of heights of all points in the cloud within search radius of the grid point, with the weights given by a Gaussian. Default search radius: max(`dem-spacing`, default_dem_spacing), so the default factor is about 1.

`--csv-format` *string*	Specify the format of input CSV files as a list of entries column_index:column_type (indices start from 1). Examples: '1:x 2:y 3:z' (a Cartesian coordinate system with origin at planet center is assumed, with the units being in meters), '5:lon 6:lat 7:radius_m' (longitude and latitude are in degrees, the radius is measured in meters from planet center), '3:lat 2:lon 1:height_above_datum', 'utm:47N 1:easting 2:northing 3:height_above_datum' (the height above datum is in meters). Can also use radius_km for column_type, when it is again measured from planet center.
`--rounding-error` *float(=*$1/2^{10}$*=0.0009765625)*	How much to round the output DEM and errors, in meters (more rounding means less precision but potentially smaller size on disk). The inverse of a power of 2 is suggested.
`--dem-hole-fill-len` *int(=0)*	Maximum dimensions of a hole in the output DEM to fill in, in pixels.
`--orthoimage-hole-fill-len` *int(=0)*	Maximum dimensions of a hole in the output orthoimage to fill in, in pixels.
`--hole-fill-mode` *int(=1)*	Choose the algorithm to fill holes. [1: Interpolate based on valid values in four directions: left, right, up, and down (fast). 2: Weighted average of all valid pixels within a window of size hole-fill-len (slow).
`--hole-fill-num-smooth-iter` *int(=4)*	How many times to iterate to smooth the result of hole-filling with a Gaussian kernel.
`--remove-outliers-params` *pct (float) factor (float) [default: 75.0 3.0]*	Outlier removal based on percentage. Points with triangulation error larger than pct-th percentile times factor will be removed as outliers.
`--max-valid-triangulation-error` *float(=0)*	Outlier removal based on threshold. Points with triangulation error larger than this (in meters) will be removed from the cloud.
`--use-surface-sampling` *[default: false]*	Use the older algorithm, interpret the point cloud as a surface made up of triangles and sample it (prone to aliasing).
`--fsaa` *float(=3)*	Oversampling amount to perform antialiasing. Obsolete, can be used only in conjunction with `--use-surface-sampling`.
`--threads` *int(=0)*	Select the number of processors (threads) to use.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.
`--cache-dir` *directory(=/tmp)*	Folder for temporary files. Normally this need not be changed.
`--help|-h`	Display the help message.

A.5 point2mesh

Produces a mesh surface that can be visualized in `osgviewer`, which is a standard 3D viewing application that is part of the open source OpenSceneGraph package. [1]

Unlike DEMs, the 3D mesh is not meant to be used as a finished scientific product. Rather, it can be used for fast visualization to create a 3D view of the generated terrain.

The `point2mesh` program requires a point cloud file and an optional texture file (*output-prefix*`-PC.tif` and normally *output-prefix*`-L.tif`). When a texture file is not provided, a 1D texture is applied in the local Z direction that produces a rough rendition of a contour map. In either case, `point2mesh` will produce a *output-prefix*`.osgb` file that contains the 3D model in OpenSceneGraph format.

Two options for `osgviewer` bear pointing out: the `-l` flag indicates that synthetic lighting should be activated for the model, which can make it easier to see fine detail in the model by providing some real-time, interactive hillshading. The `-s` flag sets the sub-sampling rate, and dictates the degree to which the 3D model should be simplified. For 3D reconstructions, this can be essential for producing a model that can fit in memory. The default value is 10, meaning every 10th point is used in the X and Y directions. In other words that mean only $1/10^2$ of the points are being used to create the model. Adjust this sampling rate according to how much detail is desired, but remember that large models will impact the frame rate of the 3D viewer and affect performance.

Example:

`point2mesh -s 2` *output-prefix*`-PC.tif` *output-prefix*`-L.tif`

To view the resulting *output-prefix*`.osgb` file use `osgviewer`.

Fullscreen:

`> osgviewer` *output-prefix*`.osgb`

In a window:

`> osgviewer` *output-prefix*`.osgb --window 50 50 1000 1000`

Inside `osgviewer`, the keys L, T, W, and F can be used to toggle on and off lighting, texture, wireframe, and full-screen modes. The left, middle, and right mouse buttons control rotation, panning, and zooming of the model.

Table A.6: Command-line options for point2mesh

Options	Description
`--help|-h`	Display the help message.
`--simplify-mesh` *float*	Run OSG Simplifier on mesh, 1.0 = 100%.
`--smooth-mesh`	Run OSG Smoother on mesh
`--use-delaunay`	Uses the delaunay triangulator to create a surface from the point cloud. This is not recommended for point clouds with noise issues.
`--step|-s` *integer(=10)*	Sampling step size for the mesher.
`--input-file` *pointcloud-file*	Explicitly specify the input file.
`--output-prefix|-o` *output-prefix*	Specify the output prefix.
`--texture-file` *texture-file*	Explicitly specify the texture file.
`--output-filetype|-t` *type(=ive)*	Specify the output file type.
`--enable-lighting|-l`	Enables shades and lighting on the mesh.

[1] The full OpenSceneGraph package is not bundled with the Stereo Pipeline, but the `osgviewer` program is. You can download and install this package separately from http://www.openscenegraph.org/.

`--center`	Center the model around the origin. Use this option if you are experiencing numerical precision issues.

A.6 dem_mosaic

The program `dem_mosaic` takes as input a list of DEM files, optionally erodes pixels at the DEM boundaries, and creates a mosaic, blending the DEMs where they overlap.

Usage:

```
dem_mosaic [options] <dem files or -l dem_files_list.txt> -o output_file_prefix
```

The input DEM can either be set on the command line, or if there are too many they can be listed in a text file (one per line) and that file can be passed to the tool.

The output mosaic is written as non-overlapping tiles with desired tile size, with the size set either in pixels or in georeferenced (projected) units. The default tile size is large enough that normally the entire mosaic is saved as one tile.

Individual tiles can be saved via the `--tile-index` option (the tool displays the total number of tiles when it is being run). As such, separate processes can be invoked for individual tiles for increased robustness and perhaps speed.

The output mosaic tiles will be named <output prefix>-tile-<tile index>.tif, where <output prefix> is an arbitrary string. For example, if it is set to `results/output`, all the tiles will be in the `results` directory. The tile names will be adjusted accordingly if one of the `--first`, `--last`, `--min`, etc. options is invoked (see below).

By the default, the output mosaicked DEM will use the same grid size and projection as the first input DEM. These can be changed via the `--tr` and `--t_srs` options.

Instead of blending, `dem_mosaic` can compute the image of first, last, minimum, maximum, mean, median, and count of all encountered valid DEM heights at output grid points. For the “first” and “last” operations, we use the order in which DEMs were passed in.

Table A.7: Command-line options for dem_mosaic

Options	Description
`-l | --dem-list-file` *string*	Text file listing the DEM files to mosaic, one per line.
`-o | --output-prefix` *string*	Specify the output prefix.
`--tile-size` *integer(=1000000)*	The maximum size of output DEM tile files to write, in pixels.
`--tile-index` *integer*	The index of the tile to save (starting from zero). When this program is invoked, it will print out how many tiles are there. Default: save all tiles.
`--erode-length` *integer(=0)*	Erode input DEMs by this many pixels at boundary and hole edges before mosaicking them.
`--blending-length` *integer(=200)*	Larger values of this number (measured in input DEM pixels) may result in smoother blending while using more memory and computing time.
`--tr` *double*	Output DEM resolution in target georeferenced units per pixel. Default: use the same resolution as the first DEM to be mosaicked.
`--t_srs` *string*	Specify the output projection (PROJ.4 string). Default: use the one from the first DEM to be mosaicked.

`--t_projwin` *xmin ymin xmax ymax*	Limit the mosaic to this region, with the corners given in georeferenced coordinates (xmin ymin xmax ymax). Max is exclusive.
`--georef-tile-size` *double*	Set the tile size in georeferenced (projected) units (e.g., degrees or meters).
`--output-nodata-value` *double*	No-data value to use on output. Default: use the one from the first DEM to be mosaicked.
`--first`	Keep the first encountered DEM value (in the input order).
`--last`	Keep the last encountered DEM value (in the input order).
`--min`	Keep the smallest encountered DEM value.
`--max`	Keep the largest encountered DEM value.
`--mean`	Find the mean DEM value.
`--median`	Find the median DEM value (this can be memory-intensive, fewer threads are suggested).
`--count`	Each pixel is set to the number of valid DEM heights at that pixel.
`--threads` *integer(=4)*	Set the number of threads to use.
`--help|-h`	Display the help message.

A.7 dem_geoid

This tool takes as input a DEM whose height values are relative to the datum ellipsoid, and adjusts those values to be relative to the equipotential surface of the planet (geoid on Earth, and areoid on Mars). The program can also apply the reverse of this adjustment. The adjustment simply subtracts from the DEM height the geoid height (correcting, if need be, for differences in dimensions between the DEM and geoid datum ellipsoids).

Three geoids and one areoid are supported. The Earth geoids are: EGM96 and EGM2008, relative to the WGS84 datum ellipsoid (http://earth-info.nga.mil/GandG/wgs84/gravitymod/egm96/egm96.html, http://earth-info.nga.mil/GandG/wgs84/gravitymod/egm2008/egm08_wgs84.html) and NAVD88, relative to the NAD83 datum ellipsoid (http://www.ngs.noaa.gov/GEOID/GEOID09/).

The Mars areoid is MOLA MEGDR (http://geo.pds.nasa.gov/missions/mgs/megdr.html). When importing it into ASP, we adjusted the areoid height values to be relative to the IAU reference spheroid for Mars of radius 3,396,190 m, to be consistent with the DEM data produced by ASP. The areoid at that source was relative to the Mars radius of 3,396,000 m.

Table A.8: Command-line options for dem_geoid

Options	Description
`--help|-h`	Display the help message.
`--nodata-value` *integer(=-32768)*	The value of no-data pixels, unless specified in the DEM.
`--geoid` *string*	Specify the geoid to use for Earth WGS84 DEMs. Options: EGM96, EGM2008. Default: EGM96.
`--output-prefix|-o` *filename*	Specify the output file prefix.
`--double`	Output using double precision (64 bit) instead of float (32 bit).
`--reverse-adjustment`	Go from DEM relative to the geoid/areoid to DEM relative to the datum ellipsoid.

A.8 dg_mosaic

This tool can be used when processing Digital Globe Imagery (chapter 4). A Digital Globe satellite may take a picture, and then split it into several images and corresponding camera XML files. `dg_mosaic` will mosaic these images into a single file, and create the appropriate combined camera XML file.

Digital Globe camera files contain, in addition to the original camera models, their RPC approximations (section 8.10). `dg_mosaic` outputs both types of combined models. The combined RPC model can be used to map-project the mosaicked images with the goal of computing stereo from them (section 4.2).

The tool needs to be applied twice, for both the left and right image sets.

`dg_mosaic` can also reduce the image resolution while creating the mosaics (with the camera files modified accordingly).

Table A.9: Command-line options for dg_mosaic

Options	Description
`--help|-h`	Display the help message.
`--reduce-percent` *integer(=100)*	Render a reduced resolution image and XML based on this percentage.

`--skip-rpc-gen` *[default: false]*	Skip RPC model generation.
`--rpc-penalty-weight` *float(=0.1)*	The weight to use to penalize higher order RPC coefficients when generating the combined RPC model. Higher penalty weight results in smaller such coefficients.
`--output-prefix` *string*	The prefix for the output .tif and .xml files.
`--band` *integer*	Which band to use (for multi-spectral images).
`--input-nodata-value` *float*	Nodata value to use on input; input pixel values less than or equal to this are considered invalid.
`--output-nodata-value` *float*	Nodata value to use on output.
`--preview`	Render a small 8 bit png of the input for preview.
`--` *dry-run|-n*	Make calculations, but just print out the commands.

A.9 mapproject

The tool `mapproject` is used to map-project a camera image onto a DEM. The obtained images can be used, for example, to visualize how camera images would look when projected onto the ground obtained by doing stereo of these images (ideally, if there were no correlation or triangulation error, the images would project perfectly). The tool can also be used to compute stereo from the obtained map-projected images; this functionality is currently supported only with RPC models (section 4.2).

`mapproject` supersedes the older `orthoproject` tool, which could map-project only with ISIS and pinhole camera models (the latter program is still being kept for a few releases for backward compatibility). We ported all features of `orthoproject` except for projecting of vector imagery (for example, RGB pixel data).

`mapproject` is single-threaded for ISIS cameras due to the limitations of ISIS. At some point this tool will be able to distribute itself using multiple processes to work around this limitation.

Example:

```
mapproject -t isis DEM.tif image.cub camera.isis_adjust \
         output-IMG.tif --ppd 256
```

Table A.10: Command-line options for mapproject

Options	Description
`--nodata-value` *float(=-32768)*	No-data value to use unless specified in the input image.
`--t_srs`	Specify the projection (PROJ.4 string). If not provided, use the one from the DEM.
`--tr` *float*	Set the output file resolution in target georeferenced units per pixel.
`--mpp` *float*	Set the output file resolution in meters per pixel.
`--ppd` *float*	Set the output file resolution in pixels per degree.
`--session-type|-t pinhole|isis|rpc`	Select the stereo session type to use for processing. Choose 'rpc' if it is desired to later do stereo with the 'dg' session.
`--t_projwin` *xmin ymin xmax ymax*	Limit the map-projected image to this region, with the corners given in georeferenced coordinates (xmin ymin xmax ymax). Max is exclusive.
`--t_pixelwin` *xmin ymin xmax ymax*	Limit the map-projected image to this region, with the corners given in pixels (xmin ymin xmax ymax). Max is exclusive.
`--bundle-adjust-prefix` *string*	Use the camera adjustment obtained by previously running bundle_adjust with this output prefix.
`--threads` *int(=0)*	Select the number of processors (threads) to use.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.
`--cache-dir` *directory(=/tmp)*	Folder for temporary files. Normally this need not be changed.
`--help|-h`	Display the help message.

A.10 disparitydebug

The `disparitydebug` program produces output images for debugging disparity images created from `stereo`. The `stereo` tool produces several different versions of the disparity map; the most important ending with extensions `*-D.tif` and `*-F.tif`. (see Appendix C for more information.) These raw disparity map files can be useful for debugging because they contain raw disparity values as measured by the correlator; however they cannot be directly visualized or opened in a conventional image browser. The `disparitydebug` tool converts a single disparity map file into two normalized TIFF image files (`*-H.tif` and `*-V.tif`, containing the horizontal and vertical, or line and sample, components of disparity, respectively) that can be viewed using any image display program.

The `disparitydebug` program will also print out the range of disparity values in a disparity map, that can serve as useful summary statistics when tuning the search range settings in the `stereo.default` file.

Table A.11: Command-line options for disparitydebug

Options	Description
`--help|-h`	Display the help message
`--input-file` *filename*	Explicitly specify the input file
`--output-prefix|-o` *filename*	Specify the output file prefix
`--output-filetype|-t` *type(=tif)*	Specify the output file type
`--float-pixels`	Save the resulting debug images as 32 bit floating point files (if supported by the selected file type)

A.11 orbitviz

Produces a Google Earth Keyhole Markup Language (KML) file useful for visualizing camera position. The input for this tool is one or more `*.cub` files.

Table A.12: Command-line options for orbitviz

Options	Description
`--help|-h`	Display the help message
`--output|-o` *filename(=orbit.kml)*	Specifies the output file name
`--scale|-s` *float(=1)*	Scale the size of the coordinate axes by this amount. Ex: To scale axis sizes up to earth size, use 3.66
`--use_path_to_dae_model|-u` *fullpath*	Use this dae model to represent camera location. *Google Sketch up can create these.*

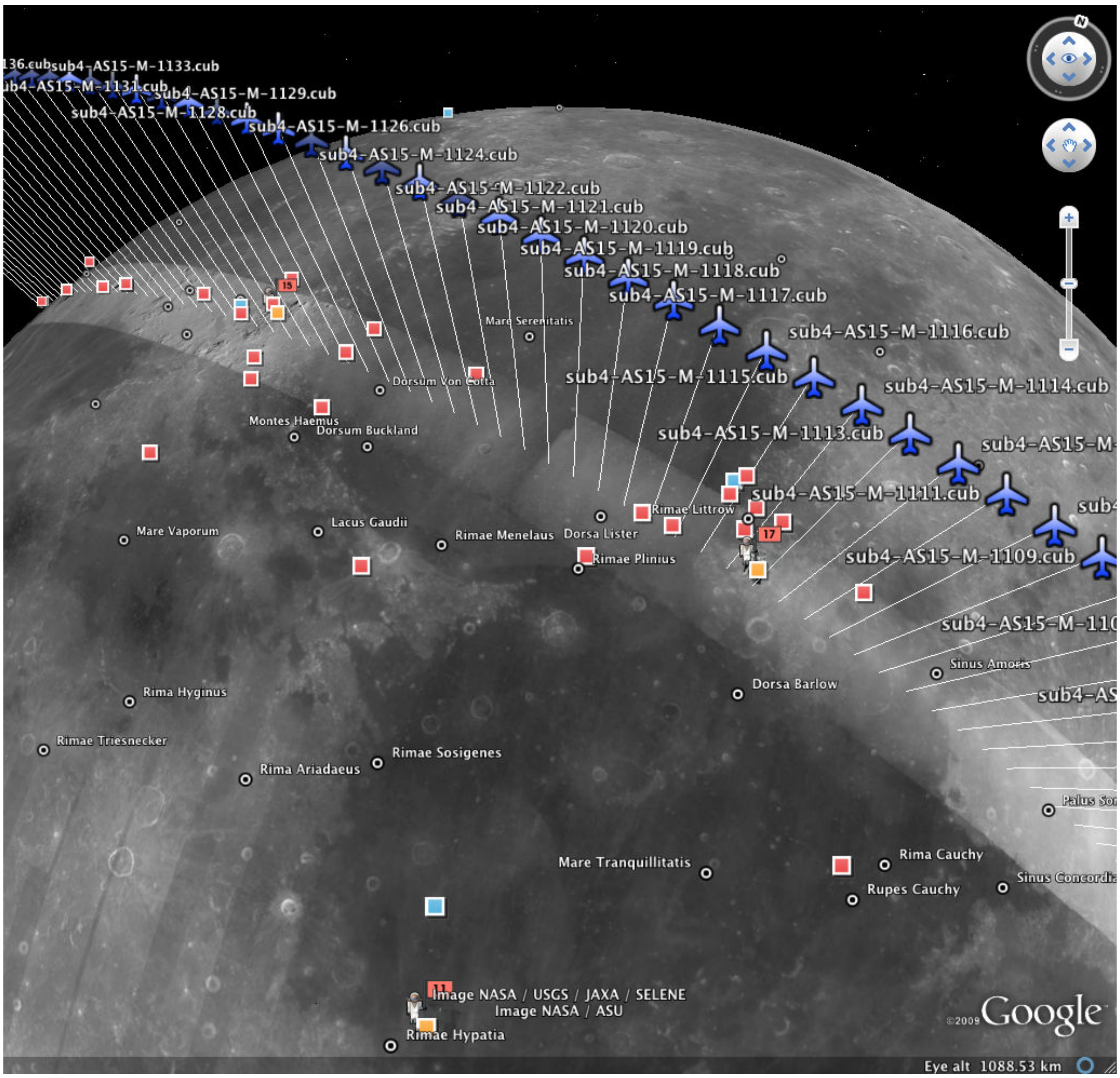


Figure A.1: Example of a KML visualization produced with `orbitviz` depicting camera locations for the Apollo 15 Metric Camera during orbit 33 of the Apollo command module.

A.12 cam2map4stereo.py

This program takes similar arguments as the ISIS3 `cam2map` program, but takes two input images. With no arguments, the program determines the minimum overlap of the two images, and the worst common resolution, and then map-projects the two images to this identical area and resolution.

The detailed reasons for doing this, and a manual step-by-step walkthrough of what `cam2map4stereo.py` does is provided in the discussion on aligning images on page 16.

The `cam2map4stereo.py` is also useful for selecting a subsection and/or reduced resolution portion of the full image. You can inspect a raw camera geometry image in qview after you have run `spiceinit` on it, select the latitude and longitude ranges, and then use `cam2map4stereo.py`'s `--lat`, `--lon`, and optionally `--resolution` options to pick out just the part you want.

Use the `--dry-run` option the first few times to get an idea of what `cam2map4stereo.py` does for you.

Table A.13: Command-line options for cam2map4stereo.py

Options	Description
`--help|-h`	Display the help message.
`--manual`	Read the manual.
`--map=`*MAP*`|-m` *MAP*	The mapfile to use for `cam2map`.
`--pixres=`*PIXRES*`|-p` *PIXRES*	The pixel resolution mode to use for `cam2map`.
`--resolution=`*RESOLUTION*`|-r` *RESOLUTION*	Resolution of the final map for `cam2map`.
`--interp=`*INTERP*`|-i` *INTERP*	Pixel interpolation scheme for `cam2map`.
`--lat=`*LAT*`|-a` *LAT*	Latitude range for `cam2map`, where `LAT` is of the form *min:max*. So to specify a latitude range between -5 and 10 degrees, it would look like `--lat=-5:10`.
`--lon=`*LON*`|-o` *LON*	Longitude range for `cam2map`, where `LON` is of the form *min:max*. So to specify a longitude range between 45 and 47 degrees, it would look like `--lon=40:47`.
`--dry-run|-n`	Make calculations, and print the `cam2map` command that would be executed, but don't actually run it.
`--suffix|-s`	Suffix that gets inserted in the output file names, defaults to 'map'.

A.13 point2las

This tool can be used to convert point clouds generated by ASP to the public LAS format for interchange of 3-dimensional point cloud data.

Table A.14: Command-line options for point2las

Options	Description
`--help|-h`	Display the help message.
`--reference-spheroid|-r earth|moon|mars`	Set the reference spheroid. This will create a geo-referenced las file in respect to the spheroid.
`--compressed`	Compress using laszip.
`--output-prefix|-o` *filename*	Specify the output file prefix.
`--threads` *integer(=0)*	Set the number threads to use. 0 means use the default defined in the program or in the .vwrc file.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.
`--cache-dir` *directory(=/tmp)*	Folder for temporary files. Normally this does not need to be changed.

A.14 pc_align

This tool can be used to align two point clouds using Point-to-Plane or Point-to-Point Iterative Closest Point (ICP). It uses the `libpointmatcher` library [25] (https://github.com/ethz-asl/libpointmatcher).

Several important things need to be kept in mind if `pc_align` is to be used successfully and give accurate results, as described below.

Due to the nature of ICP, the reference (fixed) point cloud should be denser than the source (movable) point cloud to get the most accurate results. This is not a serious restriction, as one can perform the alignment this way and then simply invert the obtained transform if desired (`pc_align` outputs both the direct and inverse transform, and can output the reference point cloud transformed to match the source and vice-versa).

In many typical applications, the source and reference point clouds are already roughly aligned, but the source point cloud may cover a larger area than the reference. The user should provide to `pc_align` the expected maximum distance (displacement) source points may move by as result of alignment, using the option `--max-displacement`. This number will help remove source points too far from the reference point cloud which may not match successfully and may degrade the accuracy. If in doubt, this value can be set to something large but still reasonable, as the tool is able to throw away a certain number of unmatched outliers. At the end of alignment, `pc_align` will display the *observed* maximum displacement, a multiple of which can be used to seed the tool in a subsequent run.

The user can choose how many points to pick from the reference and source point clouds to perform the alignment. The amount of memory and processing time used by `pc_align` is directly proportional to these numbers.

Normally Point-to-Plane ICP is more accurate than Point-to-Point, but the latter can be good enough if the input point clouds have small alignment errors and it is faster and uses less memory as well. The tool also accepts an option named `--highest-accuracy` which will compute the normals for Point-to-Plane ICP at all points rather than about a tenth of them. This option is not necessary most of the time, but may result in better alignment at the expense of using more memory and processing time.

The input point clouds can be in one of several formats: ASP's point cloud format, DEMs as GeoTIFF or ISIS cub files, LAS files, or plain-text CSV files (with .csv or .txt extension). By default, CSV files are expected to have on each line the latitude and longitude (in degrees), and the height above the datum (in meters), separated by commas or spaces with an optional header line. Alternatively, the user can specify the format of the CSV file via the `--csv-format` option. Entries in the CSV file can then be (in any order) (a) longitude, latitude (in degrees), height above datum (in meters), (b) longitude, latitude, distance from planet center (in meters or km), (c) easting, northing and height above datum in UTM coordinates (in meters), (d) Cartesian coordinates (x, y, z) measured from planet center (in meters). The precise syntax is described in the table below. The tool can also auto-detect the LOLA RDR PointPerRow format.

If none of the input files are a DEM from which the datum can be inferred, and the input files are not in Cartesian coordinates, the datum needs to be specified via the `--datum` option, or by setting `--semi-major-axis` and `--semi-minor-axis`.

The transform obtained by `pc_align` is output to a file as a 4×4 matrix with the upper-left 3×3 submatrix being the rotation and the top three elements of the right-most column being the translation. This transform, if applied to the source point cloud, will bring it in alignment with the reference point cloud. The transform assumes the 3D Cartesian coordinate system with the origin at the planet center. This matrix can be supplied back to the tool as an initial guess. The inverse transform is saved to a file as well.

The `pc_align` program outputs the translation component of this transform, defined as the vector from the centroid of the original source points to the centroid of the transformed source points. This translation component is displayed in three ways (a) Cartesian coordinates with the origin at the planet center, (b) Local North-East-Down coordinates at the centroid of the original source points, and (c) Latitude-Longitude-Height differences between the two centroids. If the effect of the transform is small (e.g., the points moved by at most several hundred meters) then the representation in the form (b) above is most amenable to interpretation as it is in respect to cardinal directions and height above ground if standing at a point on the planet surface.

The rotation + transform itself, with its origin at the center of the planet, can result in large movements on the planet surface even for small angles of rotation. Because of this it may be difficult to interpret both its rotation and translation components.

The tool outputs to CSV files the lists of errors together with their locations in the source point cloud, before and after the alignment of source points, where an error is defined as the distance from a source point used in alignment to the closest reference point. The format of output CSV files is the same as of input CSV files, or as given by `--csv-format`, although any columns of extraneous data in the input files are not saved on output.

The program prints to screen and saves to a log file the 16th, 50th, and 84th error percentiles as well as the means of the smallest 25%, 50%, 75%, and 100% of the errors.

By default, when `pc_align` discards outliers during the computation of the alignment transform, it keeps the 75% of the points with the smallest errors. As such, a way of judging the effectiveness of the tool is to look at the mean of the smallest 75% of the errors before and after alignment.

The transformed input point clouds can also be saved to disk if desired. If an input point cloud is in CSV or ASP point cloud format, the output transformed cloud will be in the same format. If the input is a DEM, the output will be an ASP point cloud, since a gridded point cloud may not stay so after a 3D transform. The `point2dem` program can be used to resample the obtained point cloud back to a DEM.

The convergence history for `pc_align` (the translation and rotation change at each iteration) is saved to disk and can be used to fine-tune the stopping criteria.

Usage:

```
pc_align  --max-displacement arg [other options] <reference cloud> <source cloud> \
  -o <output prefix>}
```

Table A.15: Command-line options for pc_align

Options	Description
`--help|-h`	Display the help message.
`--threads` *integer(=0)*	Set the number threads to use. 0 means use the default as set by OpenMP. Only some parts of the algorithm are multi-threaded.
`--initial-transform` *string*	The file containing the rotation + translation transform to be used as an initial guess. It can come from a previous run of the tool.
`--num-iterations` *default: 1000*	Maximum number of iterations.
`--diff-rotation-error` *default:* 10^{-8}	Change in rotation amount below which the algorithm will stop (if translation error is also below bound), in degrees.
`--diff-translation-error` *default:* 10^{-3}	Change in translation amount below which the algorithm will stop (if rotation error is also below bound), in meters.
`--max-displacement` *float*	Maximum expected displacement of source points as result of alignment, in meters (after the initial guess transform is applied to the source points). Used for removing gross outliers in the source (movable) point cloud.
`--outlier-ratio` *default: 0.75*	Fraction of source (movable) points considered inliers (after gross outliers further than max-displacement from reference points are removed).
`--max-num-reference-points` *default:* 10^8	Maximum number of (randomly picked) reference points to use.
`--max-num-source-points` *default:* 10^5	Maximum number of (randomly picked) source points to use (after discarding gross outliers).
`--alignment-method` *default: point-to-plane*	The type of iterative closest point method to use. [point-to-plane, point-to-point]
`--highest-accuracy`	Compute with highest accuracy for point-to-plane (can be much slower).
`--datum` *string*	Use this datum for CSV files. [WGS_1984, D_MOON (radius is assumed to be 1,737,400 meters), D_MARS (radius is assumed to be 3,396,190 meters), etc.]
`--semi-major-axis` *float*	Explicitly set the datum semi-major axis in meters.
`--semi-minor-axis` *float*	Explicitly set the datum semi-minor axis in meters.

`--csv-format` *string*	Specify the format of input (and output) CSV files as a list of entries column_index:column_type (indices start from 1). Examples: '1:x 2:y 3:z' (a Cartesian coordinate system with origin at planet center is assumed, with the units being in meters), '5:lon 6:lat 7:radius_m' (longitude and latitude are in degrees, the radius is measured in meters from planet center), '3:lat 2:lon 1:height_above_datum', 'utm:47N 1:easting 2:northing 3:height_above_datum' (the height above datum is in meters). Can also use radius_km for column_type, when it is again measured from planet center.
`--config-file` *file.yaml*	This is an advanced option. Read the alignment parameters from a configuration file, in the format expected by libpointmatcher, over-riding the command-line options.
`--output-prefix|-o` *filename*	Specify the output file prefix.
`--compute-translation-only`	Compute the transform from source to reference point cloud as a translation only (no rotation).
`--save-transformed-source-points`	Apply the obtained transform to the source points so they match the reference points and save them.
`--save-inv-transformed-reference-points`	Apply the inverse of the obtained transform to the reference points so they match the source points and save them.

A.15 wv_correct

An image taken by one of Digital Globe's World View satellite cameras is formed of several blocks as tall as the image, mosaicked from left to right, with each block coming from an individual CCD sensor [12]. Either due to imperfections in the camera or in the subsequent processing the image blocks are offset in respect to each other in both row and column directions by a subpixel amount. These so-called *CCD boundary artifacts* are not visible in the images but manifest themselves as discontinuities in the the DEMs obtained with ASP.

The tool named `wv_correct` is able to significantly attenuate these artifacts (see Figure 4.3 in the Digital Globe tutorial for an example). This tool should be used on raw Digital Globe images before calling `dg_mosaic` and `mapproject`.

It is important to note that both the positions of the CCD offsets and the offset amounts were determined empirically without knowledge of Digital Globe's mosaicking process; this is why we are not able to remove these artifacts completely.

Presently, `wv_correct` works for WV01 images for TDI of 16, 32, 48, 56 and 64, and for WV02 images for TDI of 16, 48, and 64 (both the forward and reverse scan directions for both cameras). In addition, the WV01 TDI 8 forward scan direction is supported. These are by far the most often encountered TDI. We plan to extend the tool to support other TDI when we get more such data to be able to compute the corrections.

Usage:

```
wv_correct [options] <input image> <input camera model> <output image>
```

Table A.16: Command-line options for wv_correct

Options	Description
`--help|-h`	Display the help message.
`--threads` *integer(=0)*	Set the number threads to use. 0 means use the default defined in the program or in the .vwrc file.

A.16 lronac2mosaic.py

This tool takes in two LRONAC files (M*LE.IMG and M*RE.IMG) and produces a single noproj mosaic composed of the two inputs. It performs the following operations in this process: `lronac2isis`, `lronaccal`, `lronacecho`, `spiceinit`, `noproj`, and `handmos`. The offsets used in `handmos` are calculated using an ASP internal tool called `lronacjitreg` and is similar in functionality to the ISIS command `hijitreg`. Offsets need to be calculated via feature measurements in image to correct for imperfections in camera pointing. The angle between LE and RE optics changes slightly with spacecraft temperature.

Optionally, `lronac2mosiac.py` can be given many IMG files all at once. The tool will then look at image names to determine which should be paired and mosaicked. The tool will also spawn multiple processes of ISIS commands were possible to finish the task faster. The max number of simultaneous processes is limited by the `--threads` option.

Usage:

```
lronac2mosaic.py [options] <IMG file 1> <IMG file 2>
```

Table A.17: Command-line options for lronac2mosaic.py

Options	Description
`--manual`	Display the help message.
`--output-dir|-o`	Set the output folder (default is input folder).
`--stop-at-no-proj`	Stops processing after the noproj steps are complete.
`--resume-at-no-proj`	Restarts processing using the results from 'stop-at-no-proj.
`--threads|-t`	Specify the number of threads to use.
`--keep|-k`	Keep all intermediate files.

Appendix B

The `stereo.default` File

The `stereo.default` file contains configuration parameters that the `stereo` program uses to process images. The `stereo.default` file is loaded from the current working directory when you run `stereo` unless you specify a different file using the `-s` option. Run `stereo --help` for more information. The extension is not important for this file.

A sample `stereo.default.example` file is included in the `examples/` directory of the Stereo Pipeline software distribution.

B.1 Preprocessing

---

alignment-method (= affineepipolar, homography, epipolar, none) **(default = affineepipolar)**

When `alignment-method` is set to `homography`, `stereo` will attempt to pre-align the images by automatically detecting tie-points between images using a feature based image matching technique. Tiepoints are stored in a `*.match` file that is used to compute a linear homography transformation of the right image so that it closely matches the left image. Note: the user may exercise more control over this process by using the `ipfind` and `ipmatch` tools.

When `alignment-method` is set to `affineepipolar`, `stereo` will attempt to pre-align the images by detecting tie-points, as earlier, and using those to transform the images such that pairs of conjugate epipolar lines become collinear and parallel to one of the image axes. The effect of this is equivalent to rotating the original cameras which took the pictures.

When `alignment-method` is set to `epipolar`, `stereo` will apply a 3D transform to both images so that their epipolar lines will be horizontal. This speeds of stereo correlation as it greatly reduces the area required for searching.

Epipolar alignment is only available when performing stereo matches using the pinhole stereo session (i.e. when using `stereo -t pinhole`*), and cannot be used when processing ISIS images at this time.*

force-use-entire-range (default = false)

By default, the Stereo Pipeline will normalize ISIS images so that their maximum and minimum channel values are ± 2 standard deviations from a mean value of 1.0. Use this option if you want to *disable* normalization and force the raw values to pass directly to the stereo correlations algorithms.

For example, if ISIS's `histeq` has already been used to normalize the images, then use this option to disable normalization as a (redundant) pre-processing step.

individually-normalize (default = false)

By default, the maximum and minimum valid pixel value is determined by looking at both images. Normalized with the same “global” min and max guarantees that the two images will retain their brightness and contrast relative to each other.

This option forces each image to be normalized to its own maximum and minimum valid pixel value. This is useful in the event that images have different and non-overlapping dynamic ranges. You can sometimes tell when this option is needed: after a failed stereo attempt one of the rectified images (`*-L.tif` and `*-R.tif`) may be either mostly white or black. Activating this option may correct this problem.

Note: Photometric calibration and image normalization are steps that can and should be carried out beforehand using ISIS’s own utilities. This provides the best possible input to the stereo pipeline and yields the best stereo matching results.

nodata-value (default = none)

Pixels with values less than or equal to this number are treated as no-data. This overrides the nodata values from input images.

B.2 Correlation

prefilter-mode (= 0,1,2,3) **(default = 2)**

This selects the pre-processing filter to be used to prepare imagery before it is fed to the initialization stage of the pipeline.

0 - None

1 - Subtracted Mean - This takes a preferably large Gaussian kernel and subtracts its value from the input image. This effectively reduces low frequency content in the image. The result is correlation that is immune to translations in image intensity.

2 - LoG Filter - Takes the Laplacian of Gaussian of the image, This provides some immunity to differences in lighting conditions between a pair of images by isolating and matching on blob features in the image.

3 - Sign of LoG - Not recommended for using. It was meant for an experimental XOR cost metric for correlation. This will still produce results. Though the results may not be as nice as one would like.

For all of the modes above, the size of the filter kernel is determined by the `prefilter-kernel-width` parameter below.

The choice of pre-processing filter must be made with thought to the cost function being used (see `cost-mode`, below). LoG filter preprocessing provides good immunity to variations in lighting conditions and is usually the recommended choice.

prefilter-kernel-width (= *float*) **(default = 1.4)**

This defines the diameter of the Gaussian convolution kernel used for the preprocessing modes 1 and 2 above. A value of 1.4 works well for LoG and 25-30 works well for Subtracted Mean.

corr-seed-mode (=0,1,2,3) (default = 1)

This integer parameter selects a strategy for how to solve for the low-resolution integer correlation disparity, which is used to seed the full-resolution disparity later on.

0 - None - Don't calculate a low-resolution variant of the disparity image. The search range provided by `corr-search` is used directly in computing the full-resolution disparity.

1 - Low-resolution disparity from stereo - Calculate a low-resolution version of the disparity from the integer correlation of subsampled left and right images. The low-resolution disparity will be used to narrow down the search range for the full-resolution disparity.

This is a useful option despite the fact that our integer correlation implementation does indeed use a pyramid approach. Our implementation cannot search infinitely into lower resolutions due to its independent and tiled nature. This low-resolution disparity seed is a good hybrid approach.

2 - Low-resolution disparity from an input DEM - Use a lower-resolution DEM together with an estimated value for its error to compute the low-resolution disparity, which will then be used to find the full-resolution disparity as above. These quantities can be specified via the options `disparity-estimation-dem` and `disparity-estimation-dem-error` respectively.

3 - Disparity from full-resolution images at a sparse number of points. This is an advanced option for terrain having snow and no large-scale features. It is described in section 4.4.

For large images, bigger than MOC-NA, using the low-resolution disparity seed is a definitive plus. Smaller images such as Cassini ISS or MER images should just shut this option off to save storage space.

corr-sub-seed-percent (= *float*) **(default=0.25)**
When using `corr-seed-mode 1`, the solved-for or user-provided search range is grown by this factor for the purpose of computing the low-resolution disparity.

cost-mode (= 0,1,2) (default = 2)

This defines the cost function used during integer correlation. Squared difference is the fastest cost function. However it comes at the price of not being resilient against noise. Absolute difference is the next fastest and is a better choice. Normalized cross correlation is the slowest but is designed to be more robust against image intensity changes and slight lighting differences. Normalized cross correlation is about 2x slower than absolute difference and about 3x slower than squared difference.

0 - absolute difference

1 - squared difference

2 - normalized cross correlation

corr-kernel (= *integer integer*) **(default = 25 25)**
These option determine the size (in pixels) of the correlation kernel used in the initialization step. A different size can be set in the horizontal and vertical directions, but square correlation kernels are almost always used in practice.

corr-search (= *integer integer integer integer*)
These parameters determine the size of the initial correlation search range. The ideal search range depends on a variety of factors ranging from how the images were pre-aligned to the resolution and range of disparities seen in a given image pair. This search range is successively refined during initialization, so it is often acceptable to set a large search range that is guaranteed to contain all of the disparities in a given image. However, setting tighter bounds on the search can sometimes reduce the number of erroneous matches, so it can be advantageous to tune the search range for a particular data set.

Commenting out these settings will cause `stereo` to make an attempt to guess its search range using interest points.

The order of the four integers define the minimum horizontal and vertical disparity and then the maximum horizontal and vertical disparity.

xcorr-threshold (= *integer*) **(default = 2)**

Integer correlation to a limited sense performs a correlation forward and backwards to double check its result. This is one of the first filtering steps to insure that we have indeed converged to a global minimum for an individual pixel. The `xcorr-threshold` parameter defines an agreement threshold in pixels between the forward and backward result.

Optionally, this parameter can be set to a negative number. This will signal the correlator to only use the forward correlation result. This will drastically improve speed at the cost of additional noise.

use-local-homography (**default = false**)

This flag, if provided, enables using local homography during correlation, as described in Section 6.2.2.

corr-timeout (= *integer*) **(default = 300)**

Correlation timeout for an image tile, in seconds. A non-positive value will result in no timeout enforcement.

B.3 Subpixel Refinement

subpixel-mode (= 0,1,2,3) **(default = 1)**
This parameter selects the subpixel correlation method. Parabola subpixel is very fast but will produce results that are only slightly more accurate than those produced by the initialization step. Bayes EM (mode 2) is very slow but offers the best quality. When tuning `stereo.default` parameters, it is expedient to start out using parabola subpixel as a "draft mode." When the results are looking good with parabola subpixel, then they will look even better with subpixel mode 2. For inputs with little noise, the affine method (subpixel mode 3) may produce results equivalent to Bayes EM in a shorter time.

0 - no subpixel refinement

1 - parabola fitting

2 - affine adaptive window, Bayes EM weighting

3 - affine window

4 - Lucas-Kanade method (experimental)

5 - affine adaptive window, Bayes EM with Gamma Noise Distribution (experimental)

For a visual comparison of the quality of these subpixel modes, refer back to Chapter:6.

subpixel-kernel (= *integer integer*) **(default = 35 35)** Specify the size of the horizontal and vertical size (in pixels) of the subpixel correlation kernel. It is advantageous to keep this small for parabola fitting in order to resolve finer details. However for the Bayes EM methods, keep the kernel slightly larger. Those methods weight the kernel with a Gaussian distribution, thus the effective area is small than the kernel size defined here.

B.4 Filtering

filter-mode (= *integer*) **(default = 1)**

This parameter sets the filter mode. Three modes are supported as described below. Here, by neighboring pixels for a current pixel we mean those pixels within the window of half-size of `rm-half-kernel` centered at the current pixel.

0 - No filtering.

1 - Filter by discarding pixels at which disparity differs from mean disparity of neighbors by more than `max-mean-diff`.

2 - Filter by discarding pixels at which percentage of neighboring disparities that are within `rm-threshold` of current disparity is less than `rm-min-matches`.

rm-half-kernel (= *integer integer*) **(default = 5 5)**

This setting adjusts the behavior of an outlier rejection scheme that "erodes" isolated regions of pixels in the disparity map that are in disagreement with their neighbors.

The two parameters determine the size of the half kernel that is used to perform the automatic removal of low confidence pixels. A 5×5 half kernel would result in an 11×11 kernel with 121 pixels in it.

max-mean-diff (= *integer*) **(default = 3)**

This parameter sets the *maximum difference* between the current pixel disparity and the mean of disparities of neighbors in order for a given disparity value to be retained (for `filter-mode` 1).

rm-min-matches (= *integer*) **(default = 60)**

This parameter sets the *percentage* of neighboring disparity values that must fall within the inlier threshold in order for a given disparity value to be retained (for `filter-mode` 2).

rm-threshold (= *integer*) **(default = 3)**

This parameter sets the inlier threshold for the outlier rejection scheme. This option works in conjunction with RM_MIN_MATCHES above. A disparity value is rejected if it differs by more than RM_THRESHOLD disparity values from RM_MIN_MATCHES percent of pixels in the region being considered (for `filter-mode` 2).

rm-clean-passes (= *integer*) **(default = 1)**

Select the number of outlier removal passes that are carried out. Each pass will erode pixels that do not match their neighbors. One pass is usually sufficient.

enable-fill-holes (default = false)

Enable filling of holes in disparity using an inpainting method. Obsolete. It is suggested to use instead point2dem's analogous functionality.

fill-holes-max-size (= *integer*) **(default = 100,000)**

Holes with no more pixels than this number should be filled in.

erode-max-size (= *integer*) **(default = 0)**

Isolated blobs with no more pixels than this number should be removed.

B.5 Post-Processing (Triangulation)

near-universe-radius (= *float*) **(default = 0.0)**

far-universe-radius (= *float*) **(default = 0.0)**

These parameters can be used to filter out triangulated points in the 3D point cloud. The points that will be kept are those whose distance from the universe center (see below) is between `near-universe-radius` and `far-universe-radius`, in meters.

bundle-adjust-prefix (= *string*)
Use the camera adjustments obtained by previously running bundle_adjust with this output prefix.

universe-center (default = none)
Defines the reference location to use when filtering the output point cloud using the above near and far radius options. The available options are:

None - Disable filtering.

Camera - Use the left camera's center as the universe center.

Zero - Use the center of the planet as the universe center.

point-cloud-rounding-error (= *double*)

How much to round the output point cloud values, in meters (more rounding means less precision but potentially smaller size on disk). The inverse of a power of 2 is suggested. Default: $1/2^{10}$ meters (about 1mm) for Earth and proportionally less for smaller bodies.

save-double-precision-point-cloud (default = false)

Save the final point cloud in double precision rather than bringing the points closer to origin and saving as float (marginally more precision at twice the storage).

compute-error-vector (default = false)

When writing the output point cloud, save the 3D triangulation error vector (the vector between the closest points on the rays emanating from the two cameras), rather than just its length. In this case, the point cloud will have 6 bands (storing the triangulation point and triangulation error vector) rather than the usual 4. When invoking `point2dem` on this 6-band point cloud and specifying the `--errorimage` option, the error image will contain the three components of the triangulation error vector in the North-East-Down coordinate system.

Appendix C

Guide to Output Files

The `stereo` tool generates a variety of intermediate files that are useful for debugging. These are listed below, along with brief descriptions about the contents of each file. Note that the prefix of the filename for all of these files is dictated by the final command line argument to `stereo`. Run `stereo --help` for details.

***.vwip** - image feature files

If `alignment-method` is not `none`, the Stereo Pipeline will automatically search for image features to use for tie-points. Raw image features are stored in `*.vwip` files; one per input image. For example, if your images are `left.cub` and `right.cub` you'll get `left.vwip` and `right.vwip`. Note: these files can also be generated by hand (and with finer grained control over detection algorithm options) using the `ipfind` utility.

***.match** - image to image tie-points

The match file lists a select group of unique points out of the previous `.vwip` files that have been identified and matched in a pair of images. For example, if your images are `left.cub` and `right.cub` you'll get a `left__right.match` file.

The `.vwip` and `.match` files are meant to serve as cached tie-point information, and they help speed up the pre-processing phase of the Stereo Pipeline: if these files exist then the `stereo` program will skip over the interest point alignment stage and instead use the cached tie-points contained in the `*.match` files. In the rare case that one of these files did get corrupted or your input images have changed, you may want to delete these files and allow `stereo` to regenerate them automatically. This is also recommended if you have upgraded the Stereo Pipeline software.

***-L.tif -** rectified left input image

The left input image of the stereo pair, saved after the pre-processing step. This image may be normalized, but should otherwise be identical to the original left input image.

***-R.tif -** rectified right input image

Right input image of the stereo pair, after the pre-processing step. This image may be normalized and possibly translated, scaled, and/or rotated to roughly align it with the left image, but should otherwise be identical to the original right input image.

***-lMask.tif** - mask for left rectified image

***-rMask.tif** - mask for right rectified image

These files contain binary masks for the input images. These are used throughout the stereo process to mask out pixels where there is no input data.

***-align-L.exr** - left pre-alignment matrix

***-align-R.exr** - right pre-alignment matrix

The 3×3 affine transformation matrices that are used to warp the left and right images to roughly align them. These files are only generated if `alignment-method` is not `none` in the `stereo.default` file. Normally, a single transform is enough to warp one image to another (for example, the right image to the left). The reason we use two transforms is the following: after the right image is warped to the left, we would like to additionally transform both images so that the origin (0, 0) in the left image would correspond to the same location in the right image. This will somewhat improve the efficiency of subsequent processing.

***-D.tif** - disparity map after the disparity map initialization phase

This is the disparity map generated by the correlation algorithm in the initialization phase. It contains integer values of disparity that are used to seed the subsequent sub-pixel correlation phase. It is largely unfiltered, and may contain some bad matches.

Disparity map files are stored in OpenEXR format as 3-channel, 32-bit floating point images. (Channel 0 = horizontal disparity, Channel 1 = vertical disparity, and Channel 2 = good pixel mask)

***-RD.tif -** disparity map after sub-pixel correlation

This file contains the disparity map after sub-pixel refinement. Pixel values now have sub-pixel precision, and some outliers have been rejected by the sub-pixel matching process.

***-F-corrected.tif** - intermediate data product

Only created when `alignment-method` is not `none`. This is `*-F.tif` with effects of interest point alignment removed.

***-F.tif** - filtered disparity map

The filtered, sub-pixel disparity map with outliers removed (and holes filled with the inpainting algorithm if `FILL_HOLES` is on). This is the final version of the disparity map.

***-GoodPixelMap.tif** - map of good pixels

An image showing which pixels were matched by the stereo correlator (gray pixels), and which were filled in by the hole filling algorithm (red pixels).

***-PC.tif** - point cloud image

The point cloud image is generated by the triangulation phase of Stereo Pipeline. Each pixel in the point cloud image corresponds to a pixel in the left input image (*-L.tif). The point cloud has four channels, the first three are the Cartesian coordinates of each point, and the last one has the intersection error of the two rays which created that point (the intersection error is the closest distance between rays). By default, the origin of the Cartesian coordinate system being used is a point in the neighborhood of the point cloud. This makes the values of the points in the cloud relatively small, and we save them in single precision (32 bits). This origin is saved in the point cloud as well using the tag POINT_OFFSET in the GeoTiff header. To output point clouds using double precision with the origin at the planet center, call `stereo_tri` with the option `--save-double-precision-point-cloud`. This can effectively double the size of the point cloud.

Note: it is unlikely that your usual TIFF viewing programs will visualize this file properly. This file should be considered a ‘data’ file, not an ‘image’ file. Other programs in the Stereo Pipeline, such as `point2mesh` and `point2dem` will convert the contents of this file to more easily visualized formats.

***-stereo.default** - backup of the Stereo Pipeline settings file

This is a copy of the `stereo.default` file used by `stereo`. It is stored alongside the output products as a record of the settings that were used for this particular stereo processing task.

Bibliography

[1] J. A. Anderson, S. C. Sides, D. L. Soltesz, T. L. Sucharski, and K. J. Becker. Modernization of the Integrated Software for Imagers and Spectrometers. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science XXXV*, number #2039. Lunar and Planetary Institute, Houston (CD-ROM), March 2004.

[2] J.A. Anderson. ISIS Camera Model Design. In *Proc of the Lunar and Planetary Science Conference (LPSC) XXXIX*, page 2159, March 2008.

[3] Simon Baker, Ralph Gross, and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision*, 56:221–255, 2004.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *Computer Vision and Image Understanding (CVIU)*, volume 110, pages 346–359, 2008. URL http://www.vision.ee.ethz.ch/~surf/.

[5] Michael Broxton, Ara V. Nefian, Zachary Moratto, Taemin Kim, Michael Lundy, and Aleksandr V. Segal. 3D Lunar Terrain Reconstruction from Apollo Images . In *to appear in the Proceedings of the 5th International Symposium on Visual Computing*, 2009.

[6] USGS Astrogeology Science Center. USGS ISIS Documentation. Isis 3 Application Documentation *http://isis.astrogeology.usgs.gov/Application/index.html*. URL http://isis.astrogeology.usgs.gov/Application/index.html.

[7] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14, October 2008. ISSN 0098-3500. doi: 10.1145/1391989.1391995. URL http://doi.acm.org/10.1145/1391989.1391995.

[8] The Open Scene Graph Community. The open scene graph website. 2009. URL http://www.openscenegraph.org/projects/osg.

[9] The CGIAR Consortium for Spatial Information. CGIAR-CSI SRTM 90m DEM Digital Elevation Database. URL http://srtm.csi.cgiar.org.

[10] L. Gaddis, J. Anderson, K. Becker, T. Becker, D. Cook, K. Edwards, E. Eliason, T. Hare, H. Kieffer, E. M. Lee, J. Mathews, L. Soderblom, T. Sucharski, J. Torson, A. McEwen, and M. Robinson. An Overview of the Integrated Software for Imaging Spectrometers (ISIS). In *Lunar and Planetary Science Conference*, volume 28, page 387, March 1997.

[11] GeoEye. Sample Imagery Request Form. GeoEye sample imagery request form *http://geoeye.com/CorpSite/solutions/learn-more/sample-imagery.aspx*. URL http://geoeye.com/CorpSite/solutions/learn-more/sample-imagery.aspx.

[12] Digital Globe. Radiometric Use of WorldView 2 Imagery. Description of the WV02 camera, . URL http://www.digitalglobe.com/sites/default/files/Radiometric_Use_of_WorldView-2_Imagery%20%281%29.pdf.

[13] Digital Globe. Satellite Imagery and Geospatial Information Products. Digital Globe sample imagery *http://www.digitalglobe.com/product-samples*, . URL http://www.digitalglobe.com/product-samples.

[14] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[15] Kurt Konolige. Sparse sparse bundle adjustment. In *British Machine Vision Conference*, Aberystwyth, Wales, 08/2010 2010.

[16] Daniel Machacek. Images from the long-awaited Dawn Vesta data set. http://www.planetary.org/blogs/guest-blogs/20121129-machacek-dawn-vesta.html, 2012.

[17] M. C. Malin and K. S. Edgett. Mars Global Surveyor Mars Orbiter Camera: Interplanetary cruise through primary mission. *Journal of Geophysical Research*, 106(E10):23429–23570, October 2001.

[18] M. C. Malin, G. E. Danielson, A. P. Ingersoll, H. Masursky, J. Veverka, M. A. Ravine, and T. A. Soulanille. Mars Observer Camera. *Journal of Geophysical Research*, 97(E5):7699–7718, May 1992.

[19] Christian Menard. *Robust Stereo and Adaptive Matching in Correlation Scale-Space*. PhD thesis, Institute of Automation, Vienna Institute of Technology (PRIP-TR-45), January 1997.

[20] Zach Moore, Dan Wright, Chris Lewis, and Dale Schinstock. Comparison of bundle adjustment formulations. In *ASPRS Annual Conf., Baltimore, Maryland*, 2009.

[21] Zachary Moratto. Creating control networks and bundle adjusting with isis3. http://lunokhod.org/?p=468, 2012.

[22] Zachary Moratto. Making well registered dems with isis and ames stereo pipeline. http://lunokhod.org/?p=559, 2012.

[23] Ara V. Nefian, Kyle Husmann, Michael Broxton, Mattew D. Hancher, and Michael Lundy. A Bayesian Formulation for Subpixel Refinement in Stereo Orbital Imagery. In *to appear in the Proceedings of the 2009 IEEE International Conference on Image Processing*, 2009.

[24] H.K. Nishihara. PRISM: A Practical real-time imaging stereo matcher. *Optical Engineering*, 23(5): 536–545, 1984.

[25] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP Variants on Real-World Data Sets. *Autonomous Robots*, 34(3):133–148, February 2013.

[26] Greg Slabaugh, Ron Schafer, and Mark Livingston. Optimal ray intersection for computing 3d points from n-view correspondences. http://www.soi.city.ac.uk/~sbbh653/publications/opray.pdf, 2001.

[27] Andrew Stein, Andres Huertas, and Larry Matthies. Attenuating stereo pixel-locking via affine window adaptation. In *IEEE International Conference on Robotics and Automation*, pages 914 – 921, May 2006.

[28] Changming Sun. Rectangular Subregioning and 3-D Maximum-Surface Techniques for Fast Stereo Matching. *International Journal of Computer Vision*, 47(1-3), 2002.

[29] Richard Szeliski and Daniel Scharstein. Sampling the Disparity Space Image. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26:419 – 425, 2003.

[30] Bill Triggs, Philip F. Mclauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment – a modern synthesis. *Lecture Notes in Computer Science*, 1883:298+, January 2000.

[31] AZ U.S. Geological Survey, Flagstaff. Integrated software for imagers and spectrometers (ISIS). 2009. URL http://isis.astrogeology.usgs.gov/.